Intro

introduction to machine related miscellaneous features and files

Description

The hardware-dependent section (HW) contains information useful in maintaining the system. Included are descriptions of files, devices, tables and programs that are important in maintaining the entire system that are directly related to the kind of computer on which the system runs. This section is intended for use with 80386-based computers.

80387

math coprocessor

Description

The 80387 is the INTEL math co-processor for the 80386. The kernel tests for the presence of an 80387 at startup.

If your system has an 80387, you must turn off a switch on the main system board in order to enable 80387 interrupts. Check your hardware manual to determine the proper switch and setting. If your system does not have an 80387, or the switch is on, the kernel will run a set of emulator routines which are much slower.

The C compiler available with the program development package generates the appropriate 80387 opcodes. C routines compiled with this compiler have run as much as 200 times as fast as the emulated code. In particular, the standard math library routines run considerably faster if you have an 80387.

The overflow, division by zero, and invalid operand exceptions return a SIGFPE signal. This signal can be caught. The rest of the 80387 floating point exceptions (underflow, denormalized operand, and precision error) are masked.

Notes

The emulator returns meaningless information on divide by zero.

There is no obvious way to tell which 80387 exception generated the SIGFPE.

Because of design defects in Intel's 80386 chip (B1 stepping), the Intel 80387 math co-processor may not operate correctly in some computers. The problem causes the CPU to hang when DMA/paging/coprocessor accesses are occurring. A workaround for this problem has been engineered that is engaged by using a special string at boot time:

```
Boot
: unix a31
```

Contents

Hardware Dependent **(HW)**

This workaround may not work on all machines; some hardware is designed such that it will not work. If it is successful, the following message is displayed:

```
A31 CPU bug workaround in effect
```

If unsuccessful, the following is displayed:

```
A31 CPU bug workaround not possible for this machine
```

The bootstring may also be added to the end of the default bootstring (DEFBOOTSTR) found in */etc/default/boot*.

If you cannot use this workaround, you have two options. You may replace the 386 chip with a newer release of the 386 chip (a D-step part), or you can bypass the 387 chip by adding the *ignorefpu* keyword in your boot command as follows:

```
Boot
: unix ignorefpu
```

This means that the operating system will not use the 387 chip, but you need not remove it physically; the coprocessor is still usable from DOS. To automatically bypass the 387 chip every time you boot your system, add the *ignorefpu* keyword to the */etc/default/boot* file. See **boot**(HW) for more information.

For further information, see the Intel publication: *Intel 80387 Programmer's Reference Manual*.

audit

audit subsystem interface device

Description

The audit subsystem provides two minor devices for interfacing to the audit subsystem. One device, **/dev/auditr,** is used exclusively by the audit daemon, *auditd*(ADM), for the purpose of reading the subsystem audit collection file records. The other device, **/dev/auditw,** is used by application programs which are privileged to write audit records to the audit subsystem. This device may be opened by as many applications as is necessary but may only be opened for writing. The device also support a host of *ioctl*(S) functions to perform audit subsystem control.

The audit read device provides the usual character device driver *open*(S), *read*(S), and *close*(S) routines. Writing to this device is not permitted. Read requests are satisfied by the subsystem and optimize the efficiency of the daemon and the performance of the system. Read requests are satisfied when sufficient data has accumulated to meet an administrator specified threshold. Until the data is available, the read request will block. In this manner, the daemon will receive sufficiently large blocks of data on each read to allow sufficient compaction. Also, context switch frequency is greatly reduced since the reads will not be satisfied on small blocks or when no data is available.

The audit write device provides an interface to the audit subsystem for applications that have the **writeaudit** privilege. The device supports the *openR(S), close*(S), *write*(S), and *ioctl*(S) entry points. Once opened, an application may compose an audit record and *write*(S) it to the device for inclusion in the collection file. The writing of an audit record is an atomic action in that the entire record must be presented to the subsystem with a single write. It is incumbent on the application to gather the record into a single buffer before writing it to the device.

The format of an audit record depends upon the type of event being audited. All audit records begin with a common audit record header defined by the audit_header structure in the file **sysaudit.h**.

```
struct audit_header {
    ushort      rec_length;     /* total record length */
    time_t      tstamp;         /* date/time of record */
    ulong       event_id;       /* event sequence id */
    ushort      event_type;     /* event classification */
    ushort      record_type;    /* record format */
    ushort      obj_type;       /* object type */
    ushort      pid;            /* process_id */
};
```

The *event_type*, *record_type*, and *pid* fields must be filled by the

application; all other fields are filled by the Audit subsystem. The event types are defined in the header file and provide a method of categorizing audit records into groups such as Login events or System Administrator events. The record type informs the subsystem of the record template type. This information is also retained with the record when it is written to the collection file by the subsystem since it is required at data reduction time.

Some of the record types have variable length string areas that follow the fixed portion of the audit record. Each text string that is part of the record has its size recorded in a count field. Each string is null-terminated and the count must include the null character. When the record is written to the device, the amount of data written includes the fixed portion plus all text strings. The supported record types for application programs are:

RT_LOGIN	login/logoff events
RT_PASSWORD	password modifications
RT_DATABASE	protected database modifications
RT_SUBSYSTEM	privileged subsystem events
RT_LOCK	terminal and account locking
RT_AUDIT	audit subsystem events

Each record type indicates a unique record structure definition. The **RT_LOGIN** record uses the login_audit structure. It contains the following fields, defined in **sys/audit.h**:

```
struct login_audit {
        struct audit_header aud_hdr;
        char    username[8];    /* login name */
        ushort  code;           /* function code */
        ushort  luid;           /* login userid */
        ushort  rgid;           /* real gid */
        dev_t   ttyd;           /* controlling terminal */
        ptr_t   cdir;           /* current directory */
        ptr_t   terminal;       /* stdin terminal name */
#ifdef B1
        ptr_t   sec_level;      /* login sensitivity level */
#endif
};
```

Username is the login or logoff user account name. The luid and rgid fields are those associated with the specified user account. The audit header, which precedes the login specific portion of the record, must have the *record_type* field set to **RT_LOGIN**. The *event_type* used for login/logoff is the **ET_LOGIN** event.

The *code* field is used to distinguish between specific actions that may fall into a common category. For instance, the **ET_LOGIN** event category includes both successful and unsuccessful logins, and also logoffs. The code values, defined in the header file, indicate which of these occurred.

The login audit record also contains two variable length text strings. These are the login terminal and the process current directory. The string area begins immediately following the fixed portion of the record. The size of each text string field is indicated by the ptr_t typedef field which contains the length of the string including the null character. The null character is considered part of the string. Once the strings have been calculated and the record completed, the length field in the audit record header is set to the size of structure plus the total lengths of the strings. This is the amount of data to *write*(S) to the audit device.

Modifications to user passwords are audited by the password management subsystem. Each attempt, whether successful or not, results in an audit record of type **RT_PASSWORD** being generated. The structure is defined in the *sys/sudit.h* header file:

```
struct passwd_audit {
        struct audit_header aud_hdr;
        char   username[8];  /* login user name */
        ushort code;         /* function code */
};
```

The code value distinguishes between successful and unsuccessful attempts to change the password on the indicated user account.

The system maintains a number of protected database files to support the system security policy. Attempts to modify the databases are audited with the **RT_DATABASE** type records. These records have the following format, as defined in **<sys/audit.h>**:

```
struct database_activity {
    struct audit_header aud_hdr;
    ptr_t     command;       /* command name */
    ushort    code;          /* Type of database audit */
    ushort    object;        /* object type */
    long      expected_val;  /* Expected value of parameter */
    long      present_val;   /* Present value of parameter */
    ptr_t     action;        /* security action that failed */
    ptr_t     result;        /* result of failure */
};
```

The *dbase* and *code* values identify the database and the specific action whether successful or not. A variable length text string area is provided to identify precisely what database *field* along with the *old* and *new* database field values. The audit header *length* field includes the size of the string area and the fixed portion of the record.

Protected subsystems use the **RT_SUBSYSTEM** record type to record security related events that occur in subsystem components. *Code* is used to identify the subsystem generating the record. Both the command and resulting action as well as the resulting failure are recorded in *command, action* and *result* respectively.

```
struct subsystem_activity {
    struct audit_header aud_hdr;
    ptr_t     command;     /* command name */
    ushort    code;        /* Subsystem type */
    ptr_t     action;      /* action that failed */
    ptr_t     result;      /* result of failure */
};
```

The **RT_LOCK** record type is used to audit user account and terminal locking events. The *username* identifies the user account which was locked or unlocked. *Code* distinguishes between the several events that result in the generation of a lock audit record.

```
#include<sys/audit.h>

struct lock_audit {
        struct audit_header aud_hdr;
        char    username[12];    /* login username */
        ushort  code;            /* lock function code */
        ushort  trys;            /* failed attempts */
};
```

Programs that interact with and control the audit subsystem are audited with the **RT_AUDIT** record type. The subsystem is enabled and disabled by an application program. The same is true of subsystem parameter initialization and modification. Events such as the initiation and termination of the audit daemon, the execution of the recovery mechanism, data reduction and report generation, and audit file archival are all audited.

The text string portion of the audit record is only applicable for the audit enable function since the initial subsystem collection file must be specified for the daemon log file. All other audit records do not use this field. The *code* indicates which of the above events took place.

```
struct audit_actions {
    struct audit_header aud_hdr;
    ushort    code;      /* audit function code */
    ptr_t     text1;     /* initial collection file */
};
     struct passwd_audit {
             struct audit_header aud_hdr;
             char    username[8];  /* login user name */
             ushort code;           /* function code */
     };
```

The audit device supports a number of *ioctl(S)* functions to control the audit subsystem. The format of the *ioctl(S)* calls is:

ioctl (fildes, command, arg)
int fildes, command;
struct audit_init *arg;
-or-
struct audit_ioctl *arg;
-or-
struct audit_stats *arg;

The audit_init structure is only used for **AUDIT_ENABLE** command to perform subsystem initialization. The structure is defined as follows:

```
struct audit_init {
    uint      buf_length;      /* lngth of data including header */
    mask_t    audit_flags[1];  /* audit control flags */
    mask_t    event_mask[AUDIT_MASK_SIZE];    /* system event mask */
    uint      read_count;      /* daemon read count to satisfy */
    uint      write_count;     /* write count for coll. file flush */
    long      write_time;      /* write flush time in seconds */
    long      switch_count;    /* collection file size maximum */
    long      caf_maxsize;     /* compacted audit file max size */
    uint      dir_count;       /* directory count */
    uint      uid_count;       /* uid selection count */
    uint      gid_count;       /* gid selection count */
    ulong     dir_offset;      /* fseek of directory names */
    ulong     uid_offset;      /* fseek of uids to select */
    ulong     gid_offset;      /* fseek of gids to select */
    uint      buff_count;      /* number of collection file buffers */
    ulong     session;         /* system boot session number */
    short     audit_uid;       /* audit user uid */
    short     audit_gid;       /* audit group gid */
};
```

The subsystem initialization parameters are established through the menu interface and are written to a parameter file. This file is read and used to fill out the above structure to initialize the subsystem.

The *event_mask* is a bit mask of the selected events to audit during the session. Only events that are enabled will generate audit records. The *read_count* value is used by the subsystem to satisfy audit daemon reads. Only when the specified amount of data is available in the collection file will the read be satisfied.

The flushing of the internal subsystem buffers to the collection file is controlled by the *write_count* and *write_time* fields. When the specified amount of data has accumulated, the buffers will be flushed to disk. A time interval in seconds can also be set which will cause the flushing of data to disk after a certain period of elapsed time.

The *switch_count* controls the size to which subsystem collection files may grow until a file switch is performed. The size of the output compaction files written by the audit daemon are controlled by the

AUDIT (HW)

caf_maxsize parameter. When these files reach this specified size, the daemon performs a switch to a new compaction file and records this fact in the audit session log file. *Session* is the current session value that is used in file name generation. The *buff_count* value determines the number of file system blocksize buffers to be allocated by the subsystem for the purpose of internal buffering. At least 2 buffers are allocated while 4-6 is optimal.

Dir_count is the number of collection file and compaction file directories that are available to both the subsystem and the audit daemon for the creation of their respective files. If a file write error occurs, both will attempt to use an alternate directory. Both will terminate only when all directories have been tried without success. The directory names are located in the variable length directory area following the fixed portion of the initialization record. Each pathname is a null-terminated string. The *dir_offset* field points to the start of this variable length text string area with respect to the start of the structure.

The audit subsystem is capable of selective audit record generation based on user and group IDs. These values may be specified to the subsystem at initialization time using the *uid_count* and *gid_count* values. The actual list of user and group Ids are located at the end of the structure in a variable length table of short integers. The offsets where the ID arrays may be found are located by the *uid_offset* and *gid_offset* values.

The *audit_uid* and *audit_gid* fields are used to communicate certain ID values to the subsystem since these are used to create files with specific owners and groups for security purposes.

All remaining *ioctl(S)* commands except AUDIT_STATS use the audit_ioctl structure. The audit_ioctl structure is defined by the following:

```
struct audit_ioctl {
     uint       read_count;     /* daemon read count */
     uint       write_count;    /* write count for file flush */
     long       write_time;     /* write flush time */
     mask_t     user_control[AUDIT_MASK_SIZE]; /* control mask */
     mask_t     user_disp[AUDIT_MASK_SIZE];    /* disposition mask */
     mask_t     system_mask[AUDIT_MASK_SIZE];  /* system event mask */
};
```

The AUDIT_STATS ioctl command uses the following structure for statistic retrieval and display.

```
struct audit_stats {
     uint       session;        /* current session number */
     uint       sequence;       /* current sequence number */
     ulong      total_bytes;    /* total bytes written */
     ulong      total_recs;     /* total records written */
     ulong      syscall_recs;   /* system call audit record count */
     ulong      syscall_norecs; /* system call audit record count */
     ulong      appl_recs;      /* application audit record count */
```

```
    ulong     read_count;      /* number of device reads */
    ulong     write_count;     /* number of device writes */
    ulong     coll_files;      /* number of collection files */
    ulong     buffers_used;    /* maximum audit buffer usage */
    ulong     buffer_sleep;    /* number of audit write sleeps */
};
```

The commands supported by the audit device are:

ENABLE	Initialize and enable the audit subsystem for the generation of audit records.
SHUTDOWN	Notify the audit subsystem that a system shutdown is in progress.
DISABLE	Terminate the audit subsystem and close all collection files. The audit daemon is also terminated after the last audit record has been read from the subsystem.
SYSMASK	Modify the audit subsystem event mask that controls the generation of audit records based on certain event types.
USERMASK	Modify the user event mask for a process. Each process has a mask which can be used to always or never audit certain event types regardless of the system event mask. The mask is a control mask which indicates for each bit set on that the generation of records for the corresponding event type is controlled by the second mask. The second mask is the enable/disable mask which determines whether the event is always or never audited. If a control mask bit is 0, the event is controlled by the system event mask.
FLUSH	Modify the write count and time interval values.
DAEMON	Modify the audit daemon read count value.
ACK	Used by the daemon to acknowledge certain events such as recognition of system shutdown and the disabling of the audit subsystem. Provides a synchronization means between the subsystem and the daemon.
MOUNT	The system has transitioned to multi-user state and alternate audit directories are now mounted and available.
STATS	Retrieve the current audit subsystem statistics from the audit device.

AUDIT (HW)

IDS — Specify the user and group IDs to use for selective audit generation.

Ioctl(S) calls will fail if any of the following are true:

[EPERM] The process required SelfAudit privilege but did not have it.

[EEXIST] An attempt is made to enable audit and it is already running.

[EACCES] An open attempt is made on the audit device and the calling process does not have the **configaudit** or **writeaudit** authorization.

[EBADF] *Fildes* is not a valid open file descriptor.

[EFAULT] *Arg* points to an illegal address.

[EINVAL] *Command* is an illegal value.

Files

/dev/auditr
/dev/auditw

See Also

auditd(ADM), auditcmd(ADM), "Maintaining System Security," chapter of the *System Administrator's Guide*

Diagnostics

Upon successful completion, the device returns a 0. Otherwise, a -1 is returned and *errno* is set to indicate the error.

Value Added

audit is an extension of AT&T System V provided by the Santa Cruz Operation.

boot

UNIX boot program

Description

boot is an interactive program used to load and execute stand-alone UNIX programs. It is used primarily for loading and executing the UNIX kernel, but can load and execute any other programs that are linked for stand-alone execution. *boot* is a required part of the Operating System and must be present in the root directory of the root filesystem to ensure successful loading of the UNIX kernel.

The *boot* program is invoked by the system each time the computer is started. To restart the system without going through lengthy shutdown procedures, you can use the *reboot* command. This causes the system to reboot after shutting down without waiting for keyboard input. See *haltsys*(ADM) for more information.

For diskette boot, the procedure has three stages:

1. The ROMs load the boot block from sector 0 of the floppy, where sector 0 of the disk is the same as sector 0 of the filesystem.
2. The boot block loads *boot* from the floppy filesystem.
3. *boot* executes and prompts the user.

For fixed-disk boot, the procedure has five stages:

1. The ROMs load in the *masterboot* block from sector 0 on the hard disk.
2. The *masterboot* block then loads the partition boot block (boot0) from sector 0 of the active partition (see *fdisk*(ADM)).
3. Then, assuming the UNIX partition is active, boot1 is loaded from 1K into the active partition. Boot1 spans 20 physically contiguous 1K blocks on the disk.
4. boot1 loads *boot* from the UNIX filesystem.
5. *boot* executes and prompts the user.

/boot and **/unix** may lie on tracks that have been mapped by *badtrk*(ADM). *masterboot*, boot0, and boot1 cannot lie on bad tracks.

The fixed-disk boot procedure is invoked if the diskette drive is empty.

When first invoked, *boot* prompts for the location of a program to load by displaying the message:

```
UNIX System V

Boot
:
```

To specify the location of a program, a device and filename must be given. The filename must include the full pathname of the file containing the stand-alone program. You can display a list of the current allowable device names by typing a question mark (?).

The format for the device and pathname is as follows:

xx(*m*,*o*)*filename*
or
xx(*m*)*filename*

where:
xx = device name
('hd' for the hard disk or 'fd' for diskette device)
m = minor device number
(40 for the **root** filesystem on the hard disk)
o = offset in the partition (usually 0). This is optional.
filename = standard UNIX pathname. Must start with a slash if the program is not in the root directory.

All numbers are in decimal. See the manual pages for *hd*(HW) and *fd*(HW) for minor device numbers of these devices. Specifying the offset is optional. The location of the program to be loaded must always be entered first on the command line and be present if other boot options are specified either on the command line or in **/etc/default/boot**.

If you want *boot* to pause and wait for a RETURN before executing the program that it loads, enter the word "prompt" on the command line. For example, if you enter "prompt" and press RETURN *boot* prints the following message and waits for you to press the RETURN key again:

```
Loaded, press <RETURN>.
```

The prompt can be changed to another string as in this example:

```
prompt="change diskettes now"
```

boot loads **unix** from the diskette, prints the message "change diskettes now", and waits for RETURN to be pressed. No other characters can appear between "prompt", the "=" sign, and the prompt string, although *string* may contain spaces. When you press RETURN **unix** begins execution. "Prompt" can be set either on the command

line or in **/etc/default/boot**. If a prompt is not specified, *boot* executes the loaded program without pausing.

If you have just loaded the *boot* program from the distribution diskette, simply press <RETURN> and *boot* defaults to the correct values.

To load from a hard disk, enter:

```
hd(40,0)unix
```

To use the default bootstring specified in **/etc/default/boot**, simply press <RETURN> when the system displays the boot prompt, and *boot* uses the values specified by DEFBOOTSTR in **/etc/default/boot**.

If nothing is typed after a short while and AUTOBOOT is set to YES in the default *root* filesystem's **/etc/default/boot** file, *boot* times out and behaves as though a <RETURN> had been pressed, except that an "auto" is added to the boot string. (If, in addition to AUTOBOOT=YES, TIMEOUT=*n* is defined, *boot* waits *n* seconds before timing out.) *boot* proceeds through the boot procedure, and *init*(M) is passed a **-a** flag with no "prompt".

If you wish to install DOS on the hard disk, it is recommended that you do so before you install the Operating System. See the manual page for *dos*(C). However, once you install DOS, you can boot it at the UNIX boot prompt by entering "dos".

During installation, a custom *masterboot* is placed on the hard disk. If a non-standard disk is specified, its parameters are stored and enabled in this *masterboot*.

Configuring the Kernel

boot passes any bootstring typed at the boot prompt to the kernel, except for the "prompt" string.

The kernel reads the bootstring to determine which peripherals are the root, pipe, and swap devices. If no devices are specified in either the **/etc/default/boot** description or on the command line, the default devices compiled into the kernel are used.

BOOT (HW)

Additional arguments in the bootstring can alter this default action. These arguments have the form:

dev=xx(m,o)
or
dev=xx(m)
where:
dev = the desired system device (**root[dev]**, **pipe[dev]**, or **swap[dev]**)
xx, *m*, *o* = same as for the boot device

If any combination of **root**, **pipe**, or **swap** is specified, then those system devices will reside on that device, with the unspecified system devices using the defaults compiled in the kernel. Setting one device does not affect the default values for the other system devices.

Selecting the System Console

You can select the system console at boot time either by entering the command **systty=***x* at the boot prompt, or by placing the keyword **SYSTTY=***x* in the file **/etc/default/boot**. The letter *x* represents either a number or a string parameter.

If you use the **systty=***x* command at boot time, *boot* uses the string parameter *x* to pass the selected console device to the kernel. The values of the bootstring parameter **systty** are:

sio	Serial port COM1
scrn	Display adapter

For example, to assign the system console to the serial port at COM1, enter this command at the boot prompt:

```
systty=sio
```

If you do not specifically set the system console at boot time, the *boot* program follows these steps to determine the system console:

- *boot* reads **/etc/default/boot** and looks for the keyword **SYSTTY=***x*, where *x* is a number that specifies the system console device.

  1 indicates the serial adapter at COM1
  0 indicates the display adapter

- If **SYSTTY** is not found or **/etc/default/boot** is unreadable, *boot* checks for a display adapter and assigns it as the system console.

- If no display adapter is found, *boot* looks for COM1, sets the serial port to 9600 baud, 8 data bits, 1 stop bit, and no parity, and uses it as the system console.

Thus, to have *boot* automatically set the system console to the serial port at COM1, enter this line in **/etc/default/boot**:

```
SYSTTY=1
```

Aliasing

A set of system devices can be aliased to a single keyword by defining the keyword in the file **/etc/default/boot**. This keyword can then be entered on the ''Boot'' command line and the *boot* program then reads the corresponding system devices from **/etc/default/boot** and pass them to the kernel. An alias has the following form:

key=file [root=*xx*(*m*) pipe=*xx*(*m*) swap=*xx*(*m*) prompt[="*string*"]]

In all cases, the device specification can also have the format *dev=xx*(*m*,*o*), where *o* is the offset.

For example, if you have a root file system on a second hard disk and want to use it, but want to boot using the **unix** located on the first hard disk, enter the following line into the **/etc/default/boot** description:

```
disk2=hd(40,0)unix root=hd(104,0)\
    prompt="Using second disk"
```

The next time you boot the system from the first hard disk, enter ''disk2'' in response to the ''Boot'' prompt. **unix** will be loaded from the first hard disk, and when you see the message, ''Using second disk'', press RETURN **unix** will now boot and use the root filesystem on the second hard disk. Note that you must edit the **/etc/default/boot** file in the root filesystem on the device from which *boot* will be read, in this case the first hard disk.

Another example: suppose you want to boot off the second drive (hd10) and use the root filesystem and swap space of the second drive. At the boot prompt, use the following bootstring:

```
hd(104)unix root=hd(104) pipe=hd(104)\
    swap=hd(105)
```

Once booted, you must create the device nodes for the second drive for use by the utilities:

```
fixperm -c -dHD1 /usr/lib/mkdev/perms/HD
```

Boot Options

Boot options can be changed via keywords in **/etc/default/boot**. The following keywords are recognized by *boot*:

BOOT (HW)

AUTOBOOT=YES	If YES, *boot* automatically loads UNIX after a delay time specified by the TIMEOUT parameter. The default value is 60 seconds.
DEFBOOTSTR=*string*	*string* is used as the default bootstring for timeouts or when only a <RETURN> is entered at the boot prompt. There can be no white space between DEFBOOTSTR, the "=" sign and *string*.
SYSTTY=*x*	If *x* is **1**, the system console device is set to the serial adapter at COM1. If *x* is **0** the system console is set to the main display adapter.
RONLYROOT=NO	Whether or not the root filesystem is to be mounted *readonly*. This should only be set to YES during installation.
FSCKFIX=YES or NO	Whether or not *fsck*(ADM) fixes any root system problems by itself. If the variable is set to YES, then *fsck* is run on the root filesystem with the **-rr** flag.
MULTIUSER=YES or NO	Whether or not *init*(M) invokes *sulogin* or proceeds to multiuser mode.
PANICBOOT=YES or NO	Whether or not the system reboots after a panic(). This variable is read from **/etc/default/boot** by *init*.
TIMEOUT=*n*	*n* is the number of seconds to wait at the boot prompt before timing out and booting the kernel (if AUTOBOOT is set to YES).

The following special boot options are for intended for use applications with a special need to alter init's tolerance for processes that need to be restarted.

SPAWN_INTERVAL
: The number of seconds over which "init" will try to respawn a process SPAWN_LIMIT times before it gets mad. The default value is 120.

SPAWN_LIMIT
: The number of respawns "init" will attempt in SPANW_INTERVAL seconds it generates an error message and inhibits further tries for INHIBIT seconds. The default value is 10.

SLEEPTIME
Sets the time (in seconds) between calls to *sync*.

INHIBIT
The number of seconds "init" ignores an entry it had trouble spawning unless a "telinit Q" is received. The default value is 300.

Diagnostics

If an error occurs, *masterboot* displays an error message, and locks the system. The following is a list of the most common messages and their meanings:

IO ERR An error occurred when *masterboot* tried to read in the partition boot of the active operating system.

BAD TBL The bootable partition indicator of at least one of the operating systems in the *fdisk* table contains an unrecognizable code.

NO OS There was an unrecoverable error that prevented the active operating system's partition boot from executing.

When *boot* displays error messages, it returns to the ''Boot'' prompt. The following is a list of the most common messages and their meanings:

bad magic number
The given file is not an executable program.

can't open <*pathname*>
The supplied pathname does not correspond to an existing file, or the device is unknown.

Stage 1 boot failure
The bootstrap loader cannot find or read the **boot** file. You must restart the computer and supply a filesystem disk with the **boot** file in the root directory.

not a directory
The specified area on the device does not contain a valid UNIX filesystem.

zero length directory
Although an otherwise valid filesystem was found, it contains a directory of apparently zero length. This most often occurs when a pre- System V UNIX filesystem (with incorrect, or incompatible word ordering) is in the specified area.

fload:read(*x*)=*y*
An attempted read of *x* bytes of the file returned only *y* bytes. This is probably due to a premature end-of-file. It could also be caused by a corrupted file, or incorrect word ordering in the header.

Files

/boot
/etc/default/boot
/etc/masterboot
/etc/hdboot0
/etc/hdboot1

See Also

autoboot(ADM), badtrk(ADM), fd(HW), fdisk(ADM), fsck(ADM), haltsys(ADM), hd(HW), init(M), sulogin(ADM)

Notes

The computer tries to boot off any diskette in the drive. If the diskette does not contain a valid bootstrap program, errors occur.

The *boot* program cannot be used to load programs that have not been linked for stand-alone execution. To create stand-alone programs, the **-A** option of the UNIX linker (*ld*(CP)) and special stand-alone libraries must be used.

Stand-alone programs can operate in real or protected mode, but they must not be large or huge models. Programs in real mode can use the input/output routines of the computer's startup ROM.

RONLYROOT should only be set to YES for installation. If it is set to YES during day-to-day operations, it will prevent your making changes to the root filesystem. You will then be required to boot from the floppy drive, edit the **/etc/default/boot** file, and reboot.

Value Added

boot is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

cdrom

compact disk devices

Description

The cdrom devices implement the interface with compact disk drives.

The character special cd devices (**/dev/rcd0**, and so forth) support raw I/O in multiples of the physical sector size of the CD-ROM (typically 2048 bytes).

The block special cd devices (**/dev/cd0** and so forth) support buffered I/O.

The minor device number determines which compact disk unit will be accessed. The correspondence beween the unit number and the SCSI host adaptor, controller and lun is defined in the SCSI configuration file **/etc/conf/cf.d/mscsi**.

Files

/dev/cd[0-n]
/dev/rcd[0-n]
/usr/lib/mkdev/cdrom

See Also

scsi(HW), mkdev(ADM)

Notes

Because the CD-ROM is a read-only device it is only possible to open it for input.

The command *mkdev cdrom* can be used to interactively configure the CD-ROM driver.

cmos

displays and sets the configuration data base

Syntax

cmos [address [value]]

Description

The *cmos* command displays and/or sets the values in the CMOS configuration data base. This battery-powered data base stores configuration information about the computer that is used at power up to define the system hardware configuration and to direct boot procedures. The data base is 64 bytes long and is reserved for system operation. Refer to your computer hardware manual for more information.

The *cmos* command is typically used to alter the current hardware configuration when new devices are added to the system. When only *address* is given, the command displays the value at that address. If both *address* and a *value* are given, the command assigns the value to that address. If no arguments are given, the command displays the entire contents of the data base.

The CMOS configuration data base may also be examined and modified by reading from and writing to **/dev/cmos** file. Because successful system operation depends on correct configuration information, the data base should be modified by experienced system administrators only.

The computer manufacturer's diagnostic diskette should be run before setting the CMOS data base.

Files

/etc/cmos
/dev/cmos

FD (HW)

fd

floppy devices

Description

The **fd** devices implement the interface with floppy disk drives. Each device name corresponds to a specific major and minor device. Typically, the *tar*(C), *cpio*(C) or *dd*(C) commands are used to read or write floppy disks. For instance,

```
tar tvf /dev/fd0
```

tabulates the contents of the floppy disk in drive 0 (zero).

The block special **fd** devices are also block-buffered. The floppy driver can read or write 1K bytes at a time using raw i/o. Note that block transfers are always a multiple of the 1K disk block size.

XENIX Devices

XENIX diskette device file names use the following format:

/dev/[r]fd[0,1][48ss8,48ss9,96ds9,96ds15,135ds9,135ds18]

(See Notes, below, for more information about device naming procedure.) The corresponding character special (raw) devices afford direct, unbuffered transmission between the floppy and the user's read or write transfer address in the user's program.

For information about formatting, see *format*(C).

The minor device number determines what kind of physical device is attached to each device file (see Notes). When accessing the character special floppy devices, the user's buffer must begin on a word boundary. The count in a *read*(S), *write*(S), or *lseek*(S) call to a character special floppy device must be a multiple of 1K bytes.

Device names determine the particular drive and media configuration. The device names have the form:

```
fd048ds9
```

Where:

FD (HW)

fd0 = drive number (0, 1, 2 or 3)
48 = number of disk tracks per inch (48 or 96)
ds = single or double sided floppy (ss or ds)
9 = number of sectors on the floppy (8 or 9)

For instance, /dev/fd048ss9 indicates a 48 track per inch, single sided, 9 sector floppy disk device in drive 0.

The minor device numbers for floppy drives depend on the drive and media configuration. The most common are:

	48tpi				96tpi		135tpi	
	ds/8	ds/9	ss/8	ss/9	ds/15	ds/8	ds/9	ds/18
Drive	Minor Device Number							
0	12	4	8	0	52	44	36	60
1	13	5	9	1	53	45	37	61
2	14	6	10	2	54	46	38	62
3*								

* reserved for special, non-floppy devices connected to the floppy controller as unit #3.

The scheme for creating minor device numbers is as follows. When interpreted as a binary number, each bit of the minor device number represents some aspect of the device/media configuration.

For example, the minor device number for **/dev/fd048ss8** is "8." Interpreted as a binary number, 8 is:

00001000

This is how each bit, or binary digit, is significant:

48tpi - 0 96tpi - 1 135tpi - 1	Sectors per Track		ss - 0 ds - 1	Drive	
32	16	8	4	2	1
0	0	1	0	0	0

FD (HW)

Only the last six digits of the number are used in minor device identification. The first significant digit is the third from the left. In this example, the third digit from the left is zero, thus the device is 48tpi. The next two digits mean:

Bits		Sectors per Track
16	8	
0	0	9
0	1	8
1	0	15
1	1	18

The fourth digit tells whether the floppy is single sided (ss - 0) or double sided (ds - 1). The last two signify the drive number:

Bits		Drive Number
2	1	
0	0	0
0	1	1
1	0	2
1	1	3*

* reserved for special, non-floppy devices connected to the floppy controller as unit #3.

Using this information, you can construct any minor device numbers you need.

UNIX Devices

UNIX diskette device file names use the following format:

/dev/[r]dsk/f[0,1][5h,5d9,5d8,5d4,5d16,5q,3h,3d][t,u]

where **r** indicates a raw (character) interface to the diskette, **rdsk** selects the raw device interface and **dsk** selects the block device interface. **0** or **1** selects the drive to be accessed: **f0** selects floppy drive 0, while **f1** selects drive 1. The following list describes the format to be interacted with:

5h 5.25" high density diskette (1.2MB).

5d9 5.25" double density diskette, 9 sectors per track (360KB).

5d8 5.25" double density diskette, 8 sectors per track (320KB).

5d4	5.25" double density diskette, 4 sectors per track (320KB).
5d16	5.25" double density diskette, 16 sectors per track (320KB).
5q	5.25" quad density diskette (720KB).
3h	3.50" high density diskette (1.44MB).
3d	3.50" double density diskette (720KB).

Format specification is mandatory when opening the device for formatting. However, when accessing a floppy disk for other operations (read and write), the format specification field can be omitted. In this case, the floppy disk driver will automatically determine the format previously established on the diskette and then perform the requested operation (for example, **cpio -itv</dev/rsdk/f1).**

The last parameter, **t** or **u**, selects the partition to be accessed. **t** represents the whole diskette. Without **t** or **u** specified, the whole diskette except cylinder 0 will be selected. **u** represents the whole diskette except track 0 of cylinder 0.

Besides the device file naming convention described above, some of the formats have alias names that correlate to previous releases. The following list describes the formats that have an alias:

format	alias
5h	q15d
5d8	d8d
5d9	d9d

For example, the device file **/dev/rdsk/f0q15dt** is equivalent to **/dev/rdsk/f05ht**.

Files

XENIX Devices:

/dev/[r]fd0	/dev/[r]fd048ss8	/dev/[r]fd096	/dev/[r]fd0135ds9
/dev/[r]fd1	/dev/[r]fd148ss8	/dev/[r]fd196	/dev/[r]fd1135ds9
/dev/[r]fd048	/dev/[r]fd048ds9	/dev/[r]fd096ds9	/dev/[r]fd0135ds18
/dev/[r]fd148	/dev/[r]fd148ds9	/dev/[r]fd196ds9	/dev/[r]fd1135ds18
/dev/[r]fd048ds8	/dev/[r]fd048ss9	/dev/[r]fd096ds15	
/dev/[r]fd148ds8	/dev/[r]fd148ss9	/dev/[r]fd196ds15	

FD (HW)

UNIX Devices:

/dev/[r]dsk/f0	/dev/[r]dsk/f05d9t	/dev/[r]dsk/f05d16	/dev/[r]dsk/f03ht
/dev/[r]dsk/f0t	/dev/[r]dsk/f0fd8	/dev/[r]dsk/f05d16t	/dev/[r]dsk/f03d
/dev/[r]dsk/f05h	/dev/[r]dsk/f05d8t	/dev/[r]dsk/f05q	/dev/[r]dsk/f03dt
/dev/[r]dsk/f05ht	/dev/[r]dsk/f05d4	/dev/[r]dsk/f05qt	
/dev/[r]dsk/f05d9	/dev/[r]dsk/f05d4t	/dev/[r]dsk/f03h	

Notes

It is not advisable to format a low density (48tpi) diskette on a high density (96tpi or 135tpi) floppy drive. Low density diskettes written on a high density drive should be read on high density drives. They may or may not be readable on a low density drive.

Use error-free floppy disks for best results on reading and writing.

hd

internal hard disk drive

Description

Block-buffered access to the *primary* hard disk is provided through the following block special files: **hd00**, **hd01** through **hd04**, **hd0a** and **hd0d, root**, and **swap**. Block-buffered access to the *secondary* hard disk is provided through the following block special files: **hd10**, **hd11** through **hd14**, **hd1a**.

hd00 refers to the entire physical disk; **hd01** through **hd04** refer to the fdisk partitions. **root** refers to the root file system; **swap** refers to the swap area; The block special files access the disks via the system's normal buffering mechanism and may be read and written without regard to the size of physical disk records.

Character special files follow the same naming convention as the block special files except that the character special file is prefaced with an ''r''. For example, the character special file referring to the entire physical disk is **/dev/rhd00**.

The following are the names of the fixed disk partitions. Each partition can be accessed through a block interface, for example **/dev/hd01**, or through a character (raw) interface, for example **/dev/rhd01**.

The above devices follow the XENIX naming convention. Equivalant UNIX devices are found in the **/dev/dsk** (character) and **/dev/rdsk** (raw) directories. In the table that follows, both XENIX UNIX devices are shown. XENIX devices extend only to disks located on the first controller; beyond this, the UNIX devices shown must be used.

HD (HW)

Device File Names for Fixed Disks				
First Controller		Second Controller		
Disk 1	Disk 2	Disk 1	Disk 2	Partition
/dev/hd00 /dev/rhd00	/dev/hd10 /dev/rhd10	/dev/dsk/4s0 /dev/rdsk/4s0	/dev/dsk/5s0 /dev/rdsk/5s0	entire disk
/dev/hd01 /dev/rhd01	/dev/hd11 /dev/rhd11	/dev/dsk/4s1 /dev/rdsk/4s1	/dev/dsk/5s1 /dev/rdsk/5s1	first partition
/dev/hd02 /dev/rhd02	/dev/hd12 /dev/rhd12	/dev/dsk/4s2 /dev/rdsk/4s2	/dev/dsk/5s2 /dev/rdsk/5s2	second partition
/dev/hd03 /dev/rhd03	/dev/hd13 /dev/rhd13	/dev/dsk/4s3 /dev/rdsk/4s3	/dev/dsk/5s3 /dev/rdsk/5s3	third partition
/dev/hd04 /dev/rhd04	/dev/hd14 /dev/rhd14	/dev/dsk/4s4 /dev/rdsk/4s4	/dev/dsk/5s4 /dev/rdsk/5s4	fourth partition
/dev/hd0a /dev/rhd0a	/dev/hd1a /dev/rhd1a	/dev/dsk/4sa /dev/rdsk/4sa	/dev/dsk/5sa /dev/rdsk/5sa	active partition
/dev/hd0d /dev/rhd0d	/dev/hd1d /dev/rhd1d	/dev/dsk/4sd /dev/rdsk/4sd	/dev/dsk/5sd /dev/rdsk/5sd	DOS partition
/dev/root /dev/rroot				root file system
/dev/swap /dev/rswap				swap area

Note that the root and swap file exist only for the root disk.

To access DOS partitions, specify letters such as "C:" or "D:" to indicate first or second partitions. The file **/etc/default/msdos** contains lines that assign a letter abbreviation for the DOS device name. Refer to *dos*(C).

The following table lists the minor device number definitions for the hard disk special files, along with examples. Note that the block and character special devices share the same minor device definition. The minor device number definition is as follows: bits 7 and 6 denote physical drive, bits 5-3 denote virtual(*fdisk*) partition and bits 2-0 denote *divvy* partition.

HD (HW)

Minor Device Bits				
Phys. 7 6	Virtual 5 4 3	divvy 2 1 0	Device special file name	Description
0 0	0 0 0	0 0 0	/dev/hd00	whole PD 0
0 1	0 0 0	0 0 0	/dev/hd10	whole PD 1
1 0	0 0 0	0 0 0	/dev/dsk/4s0	whole PD 2
1 1	0 0 0	0 0 0	/dev/dsk/5s0	whole PD 3
0 0	0 0 1	1 1 1	/dev/hd01	PD 0, whole VD 1
0 0	0 1 0	1 1 1	/dev/hd02	PD 0, whole VD 2
0 0	0 1 1	1 1 1	/dev/hd03	PD 0, whole VD 3
0 0	1 0 0	1 1 1	/dev/hd04	PD 0, whole VD 4
0 0	1 0 1	1 1 1	/dev/hd0a	PD 0, whole active VD
0 0	1 1 0	1 1 1	/dev/hd0d	PD 0, whole DOS VD
0 0	1 0 1	0 0 0	/dev/root	PD 0, active virtual, DP 0
0 0	1 0 1	0 0 1	/dev/swap	PD 0, active virtual, DP 1
0 0	1 0 1	0 1 0	/dev/usr	PD 0, active virtual, DP 2
0 0	1 0 1	1 1 0	/dev/recover	PD 0, active virtual, DP 6
0 1	0 0 1	1 1 1	/dev/hd11	PD 1, whole VD 1
0 1	0 1 0	1 1 1	/dev/hd12	PD 1, whole VD 2
0 1	0 1 1	1 1 1	/dev/hd13	PD 1, whole VD 3
0 1	1 0 0	1 1 1	/dev/hd14	PD 1, whole VD 4
0 1	1 0 1	1 1 1	/dev/hd1a	PD 1, whole active VD
0 1	1 1 0	1 1 1	/dev/hd1d	PD 1, whole DOS VD
0 1	1 0 1	0 0 0	/dev/u0	PD 1, active virtual, DP 0†
0 1	1 0 1	0 0 1	/dev/u1	PD 1, active virtual, DP 1†
0 1	1 0 1	0 1 0	/dev/u2	PD 1, active virtual, DP 2†
1 1	0 0 1	1 1 1	/dev/dsk/4s1	PD 2, whole VD 1
1 1	0 1 0	1 1 1	/dev/dsk/4s2	PD 2, whole VD 2
1 1	0 1 1	1 1 1	/dev/dsk/4s3	PD 2, whole VD 3
1 1	1 0 0	1 1 1	/dev/dsk/4s4	PD 2, whole VD 4
1 1	1 0 1	1 1 1	/dev/dsk/4sa	PD 2, whole active VD
1 1	1 1 0	1 1 1	/dev/dsk/4sd	PD 2, whole DOS VD
1 1	0 0 1	1 1 1	/dev/dsk/5s1	PD 3, whole VD 1
1 1	0 1 0	1 1 1	/dev/dsk/5s2	PD 3, whole VD 2
1 1	0 1 1	1 1 1	/dev/dsk/5s3	PD 3, whole VD 3
1 1	1 0 0	1 1 1	/dev/dsk/5s4	PD 3, whole VD 4
1 1	1 0 1	1 1 1	/dev/dsk/5sa	PD 3, whole active VD
1 1	1 1 0	1 1 1	/dev/dsk/5sd	PD 3, whole DOS VD

KEY	VD = virtual drive DP = divvy partition	PD = physical drive † = user-defined name

The device files **usr** and **u[0-2]** are optional filesystem names; these nodes are not present unless created by the system administrator.

HD (HW)

Files

/dev/hd0a
/dev/rhd0a
/dev/hd0[0-4]
/dev/rhd0[0-4]
/dev/hd0d
/dev/rhd0d
/dev/dsk/4s0
/dev/dsk/5s0
/dev/dsk/4s1
/dev/dsk/4s2
/dev/dsk/4s3
/dev/hd1a
/dev/rhd1a
/dev/hd1[0-4]
/dev/rhd1[0-4]
/dev/hd1d
/dev/rhd1d
/dev/dsk/4s4
/dev/dsk/4sa
/dev/dsk/4sd
/dev/dsk/5s1
/dev/dsk/5s2
/dev/usr
/dev/rusr
/dev/root
/dev/rroot
/dev/swap
/dev/rswap
/dev/dsk/5s3
/dev/dsk/5s4
/dev/dsk/5sa
/dev/dsk/5sd

See Also

fdisk(ADM), badtrk(ADM), divvy(ADM), dos(C), mkdev(ADM)

Diagnostics

The following messages are among those that may be printed on the console:

invalid fixed disk parameter table

and:

error on fixed disk (minor *n*), block = *nnnnn*,
cmd=*nnnnn*, status=*nnnn*,
Sector = *nnnnn*, Cylinder/head = *nnnnn*

Possible reasons for the first error include:

- The kernel is unable to get drive specifications, such as number of heads, cylinders, and sectors per track, from the disk controller ROM.
- Improper configuration.
- The disk is not turned on.
- The disk is not supported.

The second error specifies the following information:

- *block :* The UNIX block number within the device.

HD (HW)

- *cmd :* The last command sent to the disk controller.
- *status :* The error status from the disk controller.
- *Sector* and *Cylinder/head* specify the location of a possible flaw. This information is used with *badtrk*(ADM).

Notes

On the first disk, **hd00** denotes the entire disk and is used to access the master boot block which includes the fdisk partition table. For the second disk, **hd10** denotes the entire disk and is used to access its fdisk partition table. Do not write to **hd10** and **hd00**.

keyboard

the PC keyboard

Description

The PC keyboard is used to enter data, switch screens, and send certain control signals to the computer. The Operating System performs terminal emulation on the PC screen and keyboard, and, in doing so, makes use of several particular keys and key combinations. These keys and key combinations have special names that are unique to UNIX systems, and may or may not correspond to the keytop labels on your keyboard. These keys are described later.

When you press a key, one of the following happens:

- An ASCII value is entered
- A string is sent to the computer.
- A function is initiated.
- The meaning of another key, or keys, is changed.

When a key is pressed (a keystroke), the keyboard sends a scancode to the computer, it is interpreted by the keyboard driver. The interpretation of key codes may be modified so that keys can function differently from their default actions.

There are three special occurrences, or keystrokes:

- Switch screens.
- Send signals.
- Change the value of previous character, characters or string.

Switching Screens (Multiscreen)

To get to the next consecutive screen, enter **Ctrl-PrtSc** using the **Ctrl** key, and the **PrtSc** key. Any active screen may be selected by entering **alt-F***n*, where **F**n is one of the function keys. **F1** refers to the PC display (**/dev/tty01**).

Signals

A signal affects some process or processes. Examples of signals are **Ctrl-d** (end of input, exits from shell), **Ctrl-** (quits a process), **Ctrl-s** (stop output to the screen), and **Ctrl-q** (resume sending output).

Typically, characters are mapped to signals using *stty*(C). The only way to map signals is using *stty*.

Altering Values

The actual code sent to the keyboard driver can be changed by using certain keys in combination. For example, the SHIFT key changes the ASCII values of the alphanumeric keys. Holding down the **Ctrl** key while pressing another key sends a control code (**Ctrl-d**, **Ctrl-s**, **Ctrl-q**, etc.).

Special Keys

To help you find the special keys, the following table shows which keys on a typical console correspond to UNIX system keys. In this table, a hyphen (-) between keys means 'hold down the first key while pressing the second.'

UNIX Name	Keytop	Action
INTR	Del	Stops current action and returns to the shell. This key is also called the RUB OUT or INTERRUPT key.
BACKSPACE	←	Deletes the first character to the left of the cursor. Note that the ''cursor left'' key also has a left arrow (←) on its keytop, but you cannot backspace using that key.
Ctrl-d	Ctrl-d	Signals the end of input from the keyboard; also exits current shell.
Ctrl-h	Ctrl-h	Deletes the first character to the left of the cursor. Also called the ERASE key.
Ctrl-q	Ctrl-q	Restarts printing after it has been stopped with Ctrl-s.

KEYBOARD (HW)

UNIX Name	Keytop	Action
Ctrl-s	Ctrl-s	Suspends printing on the screen (does not stop the program).
Ctrl-u	Ctrl-u	Deletes all characters on the current line. Also called the KILL key.
Ctrl-\	Ctrl-\	Quits current command and creates a *core* file, if allowed. (Recommended for debugging only.)
ESCAPE	Esc	Special code for some programs. For example, changes from insert mode to command mode in the *vi*(C) text editor.
RETURN	(down-left arrow or ENTER)	Terminates a command line and initiates an action from the shell.
F*n*	F*n*	Function key *n*. F1-F12 are unshifted, F13-F24 are shifted F1-F12, F25-F36 are Ctrl-F1 through F12, and F37-F48 are Ctrl-Shift-F1 through F12. The next F*n* keys (F49-F60) are on the number pad (unshifted): F49 - '7' F55 - '6' F50 - '8' F56 - '+' F51 - '9' F57 - '1' F52 - '-' F58 - '2' F53 - '4' F59 - '3' F54 - '5' F60 - '0' For keys F61 through F96, see **/usr/lib/keyboard/strings**. These function keys are not available on all keyboards, but you can map other keys to represent them.

The keyboard mapping is performed through a structure defined in **/usr/include/sys/keyboard.h**. Each key can have ten states. The first eight are:

- Base
- Shift
- Ctrl
- Alt
- Ctrl-Shift
- Alt-Shift
- Alt-Ctrl
- Alt-Ctrl-Shift

There are two additional states indicated by two special bytes. The first is a ''special state'' byte whose bits indicate whether the key is ''special'' in one or more of the first eight states.

The second is one of four characters (C, N, B, O) which indicate how the lock keys affect the particular key. This is discussed further in the next section, ''Scan Codes.''

Keyboard Mode

Most keyboards normally are in a PC compatibility mode, though some can be put into a native AT keyboard mode. The UNIX utility *kbmode*(ADM) can be used to determine if a keyboard supports AT mode, and can also be used to put the keyboard into AT mode until the next time the system is rebooted. A system can also be configured to boot with the keyboard in AT mode with the *configure*(ADM) utility.

Enhanced keyboards are more fully programmable in AT mode. Also, it recognizes two control keys and an alt key.

Scan Codes

The following table describes the default contents of **/usr/lib/keyboard/keys**. The column headings are:

SCAN CODE - The scan code generated by the keyboard hardware when a key is pressed. There is no user access to the scan code generated by releasing a key.

BASE - The normal value of a key press.

SHIFT - The value of a key press when the SHIFT is also being held down.

LOCK - Indicates which lock keys affect that particular key:

- C indicates Capslock
- N indicates Numlock
- B indicates both
- O indicates locking is off

Keys affected by the lock keys C, B, or N, send the shifted value (scan code) of current state when that lock key is on. When the shift key is depressed while a lock key is also on, the key reverts (toggles) to its original state.

The other columns are the values of key presses when combinations of the CTRL, ALT and SHIFT keys are also held down.

KEYBOARD (HW)

All values, except for keywords, are ASCII character values. The keywords refer to the special function keys.

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK
0	nop	nop	nop	nop	nop	nop	nop	nop	O
1	esc	esc	nop	nop	esc	esc	nop	nop	O
2	'1'	'!'	nop	nop	'1'	'!'	nop	nop	O
3	'2'	'@'	nop	nop	'2'	'@'	nop	nop	O
4	'3'	'#'	nop	nop	'3'	'#'	nop	nop	O
5	'4'	'$'	nop	nop	'4'	'$'	nop	nop	O
6	'5'	'%'	nop	nop	'5'	'%'	nop	nop	O
7	'6'	'^'	rs	rs	'6'	'^'	rs	rs	O
8	'7'	'&'	nop	nop	'7'	'&'	nop	nop	O
9	'8'	'*'	nop	nop	'8'	'*'	nop	nop	O
10	'9'	'('	nop	nop	'9'	'('	nop	nop	O
11	'0'	')'	nop	nop	'0'	')'	nop	nop	O
12	'-'	'_'	ns	ns	'-'	'_'	ns	ns	O
13	'='	'+'	nop	nop	'='	'+'	nop	nop	O
14	bs	bs	del	del	bs	bs	del	del	O
15	ht	btab	nop	nop	ht	btab	nop	nop	O
16	'q'	'Q'	dc1	dc1	'q'	'Q'	dc1	dc1	C
17	'w'	'W'	etb	etb	'w'	'W'	etb	etb	C
18	'e'	'E'	enq	enq	'e'	'E'	enq	enq	C
19	'r'	'R'	dc2	dc2	'r'	'R'	dc2	dc2	C
20	't'	'T'	dc4	dc4	't'	'T'	dc4	dc4	C
21	'y'	'Y'	em	em	'y'	'Y'	em	em	C
22	'u'	'U'	nak	nak	'u'	'U'	nak	nak	C
23	'i'	'I'	ht	ht	'i'	'I'	ht	ht	C
24	'o'	'O'	si	si	'o'	'O'	si	si	C
25	'p'	'P'	dle	dle	'p'	'P'	dle	dle	C
26	'['	'{'	esc	esc	'['	'{'	esc	esc	O
27	']'	'}'	gs	gs	']'	'}'	gs	gs	O
28	cr	cr	nl	nl	cr	cr	nl	nl	O
29	ctrl	ctrl	ctrl	ctrl	ctrl	ctrl	ctrl	ctrl	O
30	'a'	'A'	soh	soh	'a'	'A'	soh	soh	C
31	's'	'S'	dc3	dc3	's'	'S'	dc3	dc3	C
32	'd'	'D'	eot	eot	'd'	'D'	eot	eot	C
33	'f'	'F'	ack	ack	'f'	'F'	ack	ack	C
34	'g'	'G'	bel	bel	'g'	'G'	bel	bel	C
35	'h'	'H'	bs	bs	'h'	'H'	bs	bs	C
36	'j'	'J'	nl	nl	'j'	'J'	nl	nl	C
37	'k'	'K'	vt	vt	'k'	'K'	vt	vt	C
38	'l'	'L'	np	np	'l'	'L'	np	np	C
39	';'	':'	nop	nop	';'	':'	nop	nop	O
40	'\''	'"'	nop	nop	'\''	'"'	nop	nop	O
41	'`'	'~'	nop	nop	'`'	'~'	nop	nop	O

KEYBOARD (HW)

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK
42	lshift	lshift	lshift	lshift	lshift	lshift	lshift	lshift	O
43	'\\'	'|'	fs	fs	'\\'	'|'	fs	fs	O
44	'z'	'Z'	sub	sub	'z'	'Z'	sub	sub	C
45	'x'	'X'	can	can	'x'	'X'	can	can	C
46	'c'	'C'	etx	etx	'c'	'C'	etx	etx	C
47	'v'	'V'	syn	syn	'v'	'V'	syn	syn	C
48	'b'	'B'	stx	stx	'b'	'B'	stx	stx	C
49	'n'	'N'	so	so	'n'	'N'	so	so	C
50	'm'	'M'	cr	cr	'm'	'M'	cr	cr	C
51	','	'<'	nop	nop	','	'<'	nop	nop	O
52	'.'	'>'	nop	nop	'.'	'>'	nop	nop	O
53	'/'	'?'	nop	nop	'/'	'?'	nop	nop	O
54	rshift	rshift	rshift	rshift	rshift	rshift	rshift	rshift	O
55	'*'	'*'	nscr	nscr	'*'	'*'	nscr	nscr	O
56	alt	alt	alt	alt	alt	alt	alt	alt	O
57	' '	' '	' '	' '	' '	' '	' '	' '	O
58	clock	clock	clock	clock	clock	clock	clock	clock	O
59	fkey1	fkey13	fkey25	fkey37	scr1	scr11	scr1	scr11	O
60	fkey2	fkey14	fkey26	fkey38	scr2	scr12	scr2	scr12	O
61	fkey3	fkey15	fkey27	fkey39	scr3	scr13	scr3	scr13	O
62	fkey4	fkey16	fkey28	fkey40	scr4	scr14	scr4	scr14	O
63	fkey5	fkey17	fkey29	fkey41	scr5	scr15	scr5	scr15	O
64	fkey6	fkey18	fkey30	fkey42	scr6	scr16	scr6	scr16	O
65	fkey7	fkey19	fkey31	fkey43	scr7	scr7	scr7	scr7	O
66	fkey8	fkey20	fkey32	fkey44	scr8	scr8	scr8	scr8	O
67	fkey9	fkey21	fkey33	fkey45	scr9	scr9	scr9	scr9	O
68	fkey10	fkey22	fkey34	fkey46	scr10	scr10	scr10	scr10	O
69	nlock	nlock	dc3	dc3	nlock	nlock	dc3	dc3	O
70	slock	slock	del	del	slock	slock	del	del	O
71	fkey49	'7'	'7'	'7'	'7'	'7'	'7'	'7'	N
72	fkey50	'8'	'8'	'8'	'8'	'8'	'8'	'8'	N
73	fkey51	'9'	'9'	'9'	'9'	'9'	'9'	'9'	N
74	fkey52	'-'	'-'	'-'	'-'	'-'	'-'	'-'	N
75	fkey53	'4'	'4'	'4'	'4'	'4'	'4'	'4'	N
76	fkey54	'5'	'5'	'5'	'5'	'5'	'5'	'5'	N
77	fkey55	'6'	'6'	'6'	'6'	'6'	'6'	'6'	N
78	fkey56	'+'	'+'	'+'	'+'	'+'	'+'	'+'	N
79	fkey57	'1'	'1'	'1'	'1'	'1'	'1'	'1'	N
80	fkey58	'2'	'2'	'2'	'2'	'2'	'2'	'2'	N
81	fkey59	'3'	'3'	'3'	'3'	'3'	'3'	'3'	N
82	fkey60	'0'	'0'	'0'	'0'	'0'	'0'	'0'	N
83	del	'.'	del	del	del	del	del	del	N
84	nop	nop	nop	nop	nop	nop	nop	nop	O
85	fkey11	fkey23	fkey35	fkey47	scr11	scr11	scr11	scr11	O
86	fkey12	fkey24	fkey36	fkey48	scr12	scr12	scr12	scr12	O

KEYBOARD (HW)

The following scan codes exist only for keyboards which support, and are in, native AT mode rather than PC compatibility mode.

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK
87	fkey11	fkey23	fkey35	fkey47	scr11	scr11	scr11	scr11	O
88	fkey12	fkey24	fkey36	fkey48	scr12	scr12	scr12	scr12	O
89	nop	nop	nop	nop	nop	nop	nop	nop	O
90	nop	nop	nop	nop	nop	nop	nop	nop	O
91	nop	nop	nop	nop	nop	nop	nop	nop	O
92	nop	nop	nop	nop	nop	nop	nop	nop	O
93	nop	nop	nop	nop	nop	nop	nop	nop	O
94	nop	nop	nop	nop	nop	nop	nop	nop	O
95	nop	nop	nop	nop	nop	nop	nop	nop	O
96	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	O
97	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	O
98	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	O
99	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	O
100	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	O
101	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	O
102	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	O
103	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	O
104	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	O
105	del	del	del	del	del	del	del	del	N
106	fkey54	fkey54	fkey54	fkey54	fkey54	fkey54	fkey54	fkey54	O
107	nop	nop	nop	nop	nop	nop	nop	nop	O
108	nop	nop	nop	nop	nop	nop	nop	nop	O
109	nop	nop	nop	nop	nop	nop	nop	nop	O
110	nop	nop	nop	nop	nop	nop	nop	nop	O
111	nop	nop	nop	nop	nop	nop	nop	nop	O
112	nop	nop	nop	nop	nop	nop	nop	nop	O
113	nop	nop	nop	nop	nop	nop	nop	nop	O
114	nop	nop	nop	nop	nop	nop	nop	nop	O
115	nop	nop	nop	nop	nop	nop	nop	nop	O
116	nop	nop	nop	nop	nop	nop	nop	nop	O
117	nop	nop	nop	nop	nop	nop	nop	nop	O
118	nop	nop	nop	nop	nop	nop	nop	nop	O
119	nop	nop	nop	nop	nop	nop	nop	nop	O
120	nop	nop	nop	nop	nop	nop	nop	nop	O
121	nop	nop	nop	nop	nop	nop	nop	nop	O
122	nop	nop	nop	nop	nop	nop	nop	nop	O
123	nop	nop	nop	nop	nop	nop	nop	nop	O
124	nop	nop	nop	nop	nop	nop	nop	nop	O
125	nop	nop	nop	nop	nop	nop	nop	nop	O

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK
126	nop	nop	nop	nop	nop	nop	nop	nop	O
127	nop	nop	nop	nop	nop	nop	nop	nop	O
128	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	O
129	ralt	ralt	ralt	ralt	ralt	ralt	ralt	ralt	O
130	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	O
131	del	del	del	del	del	del	del	del	N
132	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	O
133	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	O
134	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	O
135	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	O
136	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	O
137	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	O
138	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	O
139	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	O
140	'/'	nop	nop	nop	'/'	nop	nop	nop	O
141	cr	cr	nl	nl	cr	cr	nl	nl	O

The next table lists the ''value'' of each of the special keywords used in **/usr/lib/keyboard/keys** (and the preceding table). *mapkey*(M) places a ''value'' in the *ioctl* buffer during key mapping. The keywords are only used in the scan code file (**/usr/lib/keyboard/keys**) for readability.

Name	Value	Meaning
nop	0	No operation - no action from keypress
lshift	2	Left hand shift
rshift	3	Right hand shift
clock	4	Caps lock
nlock	5	Numeric lock
slock	6	Scroll lock
alt	7	Alt key
btab	8	Back tab key - generates fixed sequence (esc [Z)
ctrl	9	Control key
nscr	10	Switch to the next screen
scr1	11	Switch to screen #1
...		...
scr16	26	Switch to screen #16
fkey1	27	Function key #1
...		...
fkey96	122	Function key #96
rctl	128*	Right Control Key
ralt	129*	Right Alt Key

* AT mode keyboard only.

This table lists names and decimal values that are interchangeable in the *mapkey* file. Names are used in place of numeric constants to make it easier to read the scan code table. Again, only the decimal values are placed in the *ioctl* buffer. These are taken from **ascii**(M).

Name	Value	Name	Value
nul	0	dc1	17
soh	1	dc2	18
stx	2	dc3	19
etx	3	dc4	20
eot	4	nak	21
enq	5	syn	22
ack	6	etb	23
bel	7	can	24
bs	8	em	25
ht	9	sub	26
nl	10	esc	27
vt	11	fs	28
np	12	gs	29
cr	13	rs	30
so	14	ns	31
si	15	del	127
dle	16		

Keyboard Mapping

The PC keyboard is mapped as part of terminal emulation. This kind of mapping is performed only on the computer keyboard, not on remote terminals. Use *mapkey* to change keyboard mapping. To change the mapping for individual channels (multiscreens), use *mapchan*(M).

Keyboard mapping can also be performed using *ioctl*. The syntax is the same as for string key mapping (see previous section).

For keyboard mapping, *cmd* is GIO_KEYMAP to display the current map, and PIO_KEYMAP puts the prepared buffer into place.

String Key Mapping

To map string (function) keys, use the *mapstr* (see *mapkey*(M)) utility. *mapstr* modifies the string mapping table where function keys are defined.

The string mapping table is an array of 512 bytes (typedef *strmap_t*) containing null terminated strings that redefine the function keys. The first null terminated string is assigned to the first string key, the second string to the second string key, and so on.

There is no limit to the length of any particular string as long as the whole table does not exceed 512 bytes, including nulls. Strings are made null by the introduction of extra null characters.

The following is a list of default function key values:

Default Function Key Values		
Key Number	Function Key	Function
1	F1	ESC [M
2	F2	ESC [N
3	F3	ESC [O
4	F4	ESC [P
5	F5	ESC [Q
6	F6	ESC [R
7	F7	ESC [S
8	F8	ESC [T
9	F9	ESC [U
10	F10	ESC [V
11	F11	ESC [W
12	F12	ESC [X
13	Shift-F1	ESC [Y
14	Shift-F2	ESC [Z
15	Shift-F3	ESC [a
16	Shift-F4	ESC [b
17	Shift-F5	ESC [c
18	Shift-F6	ESC [d
19	Shift-F7	ESC [e
20	Shift-F8	ESC [f
21	Shift-F9	ESC [g
22	Shift-F10	ESC [h
23	Shift-F11	ESC [i
24	Shift-F12	ESC [j
25	Ctrl-F1	ESC [k
26	Ctrl-F2	ESC [l
27	Ctrl-F3	ESC [m
28	Ctrl-F4	ESC [n
29	Ctrl-F5	ESC [o
30	Ctrl-F6	ESC [p
31	Ctrl-F7	ESC [q
32	Ctrl-F8	ESC [r
33	Ctrl-F9	ESC [s
34	Ctrl-F10	ESC [t
35	Ctrl-F11	ESC [u
36	Ctrl-F12	ESC [v

KEYBOARD (HW)

Default Function Key Values (continued)

Key Number	Function Key	Function
37	Ctrl-Shift-F1	ESC [w
38	Ctrl-Shift-F2	ESC [x
39	Ctrl-Shift-F3	ESC [y
40	Ctrl-Shift-F4	ESC [z
41	Ctrl-Shift-F5	ESC [@
42	Ctrl-Shift-F6	ESC [[
43	Ctrl-Shift-F7	ESC [\
44	Ctrl-Shift-F8	ESC []
45	Ctrl-Shift-F9	ESC [^
46	Ctrl-Shift-F10	ESC [_
47	Ctrl-Shift-F11	ESC [`
48	Ctrl-Shift-F12	ESC [{
49	Home	ESC [H
50	Up arrow	ESC [A
51	Page up	ESC [I
52	Minus sign	-
53	Left arrow	ESC [D
54	5	ESC [E
55	Right arrow	ESC [C
56	Plus sign	+
57	End	ESC [F
58	Down arrow	ESC [B
59	Page down	ESC [G
60	Insert	ESC [L

You can also map string keys using *ioctl* (S). The syntax is:

```
#include <sys/keyboard.h>
ioctl(fd,cmd,buf)
int fd, cmd;
char *buf;
...
```

For string key mapping where *cmd* is GIO_STRMAP to display the string mapping table and PIO_STRMAP to put the new string mapping table in place.

Files

/usr/lib/keyboard/keys
/usr/lib/keyboard/strings

See Also

mapchan(F), mapchan(M), mapkey(M), multiscreen(M), screen(HW), setkey(C), stty(C), kbmode(ADM), configure(ADM)

lp, lp0, lp1, lp2

line printer device interfaces

Description

The **lp0** , **lp1**, and **lp2** files provide access to the optional parallel ports of the computer. The **lp0** and **lp2** files provide access to parallel ports 1 and 2, respectively. The **lp1** file provides access to the parallel port on the monochrome adaptor.

Only one of **lp0** and **lp1** may be used on a given system. To access two parallel printers on a system, use either **lp0** or **lp1**, and **lp2**.

Files

/dev/lp0
/dev/lp1
/dev/lp2

See Also

lp(C), lpadmin(ADM), lpsched(ADM)

Notes

The standard **lp** ports, **lp0**, **lp1**, and **lp2** send a printer initialization string the first time the file is opened after the system is *booted*.

Not all computers have an alternate parallel port slot.

machine

description of host machine

Description

This page lists the internal characteristics of personal computers which use the Intel 8086 processor family and its associated hardware. The information is intended for software developers who wish to transfer relocatable object or executable files from other machines to a personal computer then prepare the files for execution on the personal computer.

Central Processing Unit	Intel 80386
Disk Block Size (BSIZE)	1024 bytes
Memory Management Scheme	Segmented and paged (80386)
Split Instruction and Data	Supported
Variable Stack Size	Supported
Fixed Stack Size	Supported
Clock Ticks	.02 second

Binary Compatibility

The small and middle model binary programs created by the C compiler *cc*(CP) run on many processors. The following chart shows which XENIX systems running on which processors produce code executable on other machines. It is assumed that system specific system calls are not used to produce portable code. *cc*(CP) produces code by default, but can also be used as a cross development compiler, by using the appropriate flags.

SCO-*nn* is XENIX distributed by The Santa Cruz Operation, Inc. MS-*nn* is XENIX distributed by Microsoft Corporation. Intel UNIX is distributed by Intel Corporation. Altos UNIX is distributed by Altos Computer Systems. *nn* designates the machine processor. System designates the version of XENIX, either 2.3, 3.0, or System V.

MACHINE (HW)

Binary Compatibility			
Your System Processor	**Default compiler produces programs which run on System/Processor**	**Runs default programs created on System/Processor**	**Compiles (cross development) programs for System/Processor**
SCO-86 3.0	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 SysV	SCO-86 3.0 SCO-186 3.0 Intel, Altos-86 2.3, 3.0	DOS*
SCO-86 SystemV	SCO-86 Sys V SCO-186 Sys V SCO-286 Sys V MS-286 Sys V	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 Sys V Intel, Altos-86 2.3, 3.0	MS-286 3.0† DOS*
SCO-186 3.0	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 Sys V	SCO-86 3.0 SCO-186 3.0 Intel, Altos-86 2.3, 3.0	DOS*
SCO-186 System V	SCO-86 Sys V SCO-186 Sys V SCO-286 Sys V MS-286 Sys V	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 Sys V Intel, Altos-86 2.3, 3.0	MS-286 3.0† DOS*
SCO-286 3.0	SCO-286 [3.0, Sys V] MS-286 [3.0†, Sys V]	SCO-286 3.0 MS-286 3.0†	DOS*
SCO-286 System V	SCO-86 Sys V SCO-186 Sys V SCO-286 Sys V MS-286 Sys V	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 [3.0, Sys V] MS-286 [3.0†, Sys V]	SCO-286 3.0 MS-286 3.0† DOS*
SCO-386 System V	SCO-86 Sys V SCO-186 Sys V SCO-286 Sys V SCO-386 Sys V MS-286 Sys V MS-386 Sys V	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 [3.0, Sys V] SCO-386 [Sys V] MS-286 [3.0†, Sys V] MS-386 [Sys V]	SCO-286 3.0 MS-286 3.0† DOS*
MS-286 3.0†	MS-286 [3.0†, Sys V] SCO-286 Sys V	SCO-286 3.0	DOS*
MS-286 System V	MS-286 Sys V SCO-286 Sys V	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 [3.0, Sys V]	DOS*
MS-386 System V	MS-386 Sys V SCO-386 Sys V	SCO-86 [3.0, Sys V] SCO-186 [3.0, Sys V] SCO-286 [3.0, Sys V] SCO-386 [Sys V]	DOS*

* MS-DOS for i8086/8088, i80186 and i80286 processors.
† MS-286 3.0 XENIX is equivalent to Intel 286 3.0 XENIX.

See Also

cc(CP), ld(CP), a.out(F)

MOUSE (HW)

mouse

system mouse

Description

UNIX supports mice attached directly to controller cards on the bus and mice attached to standard serial ports. The command:

mkdev mouse

is used to configure a new mouse or to reconfigure an existing mouse.

See Also

Files

/dev/mouse	Directory for mouse-related special device files.
/dev/mouse/bus[0-1]	Bus mouse device files.
/dev/mouse/vpix[0-1]	vpix-mouse device files.
/dev/mouse/microsoft_ser	Microsoft serial mouse device files.
/dev/mouse/logitech_ser	Logitech serial mouse device files.
/dev/mouse/mousesys_ser	Mousesys serial mouse device files.
/dev/mouse/ttyp[0-7]	Special pseudo-tty files for mouse input.
/etc/default/usemouse	Default map file for mouse-generated characters
/usr/lib/event/devices	File containing device information for mice.
/usr/lib/event/ttys	File listing ttys eligible to use mice.
/usr/lib/mouse/*	Alternate map files for mice.

mkdev(ADM), usemouse(C)

Value Added

mouse is an extension of AT&T System V provided by the Santa Cruz Operation.

parallel

parallel interface devices

Description

There are several parallel devices:

/dev/lp0 Main parallel adapter.

/dev/lp1 Adapter on monochrome video card.

/dev/lp2 Alternate parallel adapter (on appropriate machines).

It is not possible to have all three parallel devices on one system. Some AT computers allow the use of two parallel devices, **/dev/lp2**, and either **/dev/lp0** or **/dev/lp1**. However, available devices vary from machine to machine, and may instead allow either **/dev/lp0**, or and either **/dev/lp1**, and **/dev/lp2**.

If a parallel device fails to interrupt properly, the parallel driver enters ''poll mode.'' Once interrupts are received from the device, the driver returns to its original mode.

The parallel driver delays a certain amount of time when a parallel device is closed. The amount of delay can affect printer performance, but is necessary to compensate for different sizes of printer buffers and printer speeds. For example, this command sets the delay on close to 1 second, specified in 10ths of a second:

```
stty time 10< /dev/lp0
```

When given from a prompt, this command will only work if the port is open. It is recommended that a variation of this command be placed in the interface script used with the parallel device to achieve the same results:

```
stty time 10 0< &1
```

Notes

Parallel adapters on add-on cards will function, but switches must be set correctly. Some compatible computers have ports lp0 and lp1 reversed.

The *stty*(C) command for output processing is supported on a parallel device. *stty* options that have no effect on a parallel device are ignored and no error messages are displayed.

Usage

Usually invoked by *lp*(C), but can be written to directly.

Files

/dev/lp0
/dev/lp1
/dev/lp2

See Also

lp(C), lp(HW), lpadmin(ADM), lpsched(ADM), serial(HW)

prf

operating system profiler

Description

The special file */dev/prf* provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file */dev/prf* is a pseudo-device with no associated hardware.

Files

/dev/prf

See Also

profiler(ADM)

RAMDISK (HW)

ramdisk

memory block device

Description

The *ramdisk* device driver provides a block interface to memory. A *ramdisk* can be used like any other block device, including making it into a file systems using *mkfs*(ADM). There are eight *ramdisk*s available.

The characteristics of a *ramdisk* file are determined by its minor device number. The bits in the minor device number encode its size, longevity, and which of the eight possible *ramdisk*s it is.

The three low-order bits of the minor device number determine which of the eight *ramdisk*s is being accessed.

The next four bits of the minor device number determine the size of the *ramdisk*. The size of a *ramdisk* must be a power of 2, and must be at least 16K. Since 4 bits are available, there are 16 possible sizes, starting at 16K and doubling every time the size indicator is incremented, to produce possible sizes of 16K, 32K, 64K, and up.

The high-order bit is a longevity indicator. If set, memory is permanently allocated to that *ramdisk*, and can be deallocated only by rebooting the system. Permanent *ramdisk*s can only be allocated by the superuser. However, once a permanent *ramdisk* is allocated (by opening it), it can be read and written by anyone having the appropriate permissions on the *ramdisk* inode.

If clear, the *ramdisk* is deallocated when no processes have it open. To create an easily removable, but semi-permanent *ramdisk*, use a separate process to keep the device open for as long as necessary.

Since a complete set of *ramdisk*s (8) would consume 256 inodes, only one example 16K *ramdisk* (**/dev/ram00**) is created when the system is installed. The system administrator can check this existing file to determine the major device number for any other required *ramdisk*s. All *ramdisk*s will use the same major device number.

The following table shows how the minor device number is constructed:

RAMDISK (HW)

Example Minor Device Number Construction										
Description	Longe-vity	Size (see next table)				Ram Disk No.			Minor Device Number
16K (#1) (Temporary)	0	0	0	0	0	0	0	1	1
16K (#1) (Permanent)	1	0	0	0	0	0	0	1	129
64K (#0) (Temporary)	0	0	0	1	0	0	0	0	16
512K (#7) (Permanent)	1	0	1	0	1	1	1	1	175

The contents of the size field and the corresponding *ramdisk* size is shown in the next table.

Size Bits				Ramdisk Size
0	0	0	0	16K
0	0	0	1	32K
0	0	1	0	64K
0	0	1	1	128K
0	1	0	0	256K
0	1	0	1	512K
0	1	1	0	1M
0	1	1	1	2M
1	0	0	0	4M
1	0	0	1	8M
1	0	1	0	16M
1	0	1	1	32M
1	1	0	0	64M
1	1	0	1	128M
1	1	1	0	256M
1	1	1	1	512M

To create a ramdisk, follow these steps:

1. Create the device node.

You must first create the device that the ramdisk will reside on. It has the form:

mknod *device_name b_or_c major_device_number minor_device_number*

where *b_or_c* "b" or "c". "b" is for blocked devices and is the one you will use. The major number will always be 31. The minor number is derived from the table above. The minor number is the sum of the three attribute columns.

RAMDISK (HW)

Longevity:
permanent = 128 non-permanent = 0

Size:

16K = 0	128K = 24	1 Meg = 48	8 Meg = 72
32K = 8	256K = 32	2 Meg = 56	16 Meg = 80
64K = 16	512K = 40	4 Meg = 64	32 Meg = 88

Ram Disk number: 0 through 7 Note: There are only 8 devices available. Two different size devices may not share the same number.

For example, to create a 64K permanent ramdisk, the minor number could vary from 144 to 151. If the disk number was 1 the mknod command would be:

```
mknod /dev/ram64 b 31 145
```

2. Make a file system.

This creates a file system on the the ramdisk. In this example *mkfs* has the form:

mkfs *device_name size_of_file_in_Bsize_blocks*

In this example, the command to create a 64K file system would be:

```
mkfs /dev/ram64 64
```

3. Mount the filesystem.

This mounts the selected device on the specified mount point. It has the form:

```
mount device_name mount_point
```

In order to mount the example 64K ramdisk on /mnt the command would be:

```
mount /dev/ram64 /mnt
```

To make a file system on a non-permanent *ramdisk*, the device file must be held open between the *mkfs* and the *mount*(ADM) operations. Otherwise, the *ramdisk* is allocated at the start of the *mkfs* command, and deallocated at its end. Once the *ramdisk* is mounted, it is open until it is unmounted.

The following shell fragment shows one way to use *mkfs* on a non-permanent 512K *ramdisk*, then mount it:

```
(       /etc/mkfs /dev/ram40 512
        /etc/mount /dev/ram40 /mnt
) < /dev/ram40
```

Notes

*ramdisk*s must occupy contiguous memory. If free memory is fragmented, opening a *ramdisk* may fail even though there is enough total memory available. Ideally, all *ramdisk*s should be allocated at system startup. This helps prevent the *ramdisk*s themselves from fragmenting memory.

ramdisks are geared towards use in specialized applications. In many cases, you will notice a *decrease in system performance* when *ramdisks* are used, because UNIX can generally put the memory to better use elsewhere.

Files

/dev/ram00

See Also

mkfs(ADM), mount(ADM), mknod(C)

Value Added

ramdisk is an extension of AT&T System V provided by the Santa Cruz Operation.

RTC (HW)

rtc

real time clock interface

Description

The **rtc** driver supports the real time clock chip, allowing it to be set with the correct local time and allowing the time to be read from the chip.

Ioctl Calls

RTCRTIME

This call is used to read the local time from the real time clock chip. The argument to the *ioctl* is the address of a buffer of **RTCNREG** unsigned characters (**RTCNREG** is defined as **<sys/rtc.h>**). The *ioctl* will fill in the buffer with the contents of the chip registers. Currently, **RTCNREG** is 14, and the meanings of the byte registers are as follows:

Register	Contents
0	Seconds
1	Second alarm
2	Minutes
3	Minute alarm
4	Hours
5	Hour alarm
6	Day of week
7	Date of month
8	Month
9	Year
A	Status register A
B	Status register B
C	Status register C
D	Status register D

For further information on the functions of these registers, see your hardware technical reference manual.

RTCSTIME

This call is used to set the time into the real time clock chip. The argument to the *ioctl* is the address of a buffer of **RTCNREGP** unsigned characters (**RTCNREGP** as defined in **<sys/rtc.h>**). These bytes should be the desired chip register contents. Currently, **RTCNREGP** is 10, representing registers 0-9 as shown above. Note that only the super-user may open the real time clock device for writing and that the **RTCSTIME** *ioctl* will fail for any other than the super-user.

Files

/dev/rtc

screen

tty[01-n], color, monochrome, ega, vga display adapter and video monitor

Description

The **tty[01-*n*]** device files provide character I/O between the system and the video display monitor and keyboard. Each file corresponds to a separate teletype device. Although there is a maximum of 12 screens, the exact number available (*n*) depends upon the amount of memory in the computer. The screens are modeled after a 25 line, 80 column ASCII terminal, unless specified otherwise.

System error messages from the kernel are written to **/dev/console**, which is normally the current multiscreen. If the **/dev/console** is the default output device for system error messages, and the display being used is switched to graphics mode, console messages are not displayed. When the video device returns to text mode, a notice message is displayed and the text of the kernel error can be recovered from **usr/adm/messages.**

Although all **tty**[01-*n*] devices may be open concurrently, only one of the corresponding devices can be active at any given time. The active device displays its own screen and takes sole possession of the keyboard. It is an error to attempt to access the **color**, **monochrome**, or **ega** file when no corresponding adapter exists or no multiscreens are associated with it.

To get to the next consecutive screen, enter **Ctrl-PrtSc** using the **Ctrl** key, and the **PrtSc** key. Any active screen may be selected by entering **alt-Fn**, where **Fn** is one of the function keys. For example, **F1** refers to the **tty01** device.

Control Modes

Multiscreens can be reassigned to different adapters (in multi-adapter systems) with these *ioctl*s :

SWAPMONO	Selects the monochrome display as the output device for the multiscreen.
SWAPCGA	Selects the regular color display as the output device for the multiscreen.

SWAPEGA	Selects the enhanced color display as the output device for the multiscreen.
SWAPVGA	Selects the video graphics array color display as the output device for the multiscreen.

To find out which display adapter type is currently attached to the multiscreen, you can use *ioctl*(S) with the following request:

CONS_CURRENT	Returns the display adapter type currently associated with the multiscreen. The return value can be one of: MONO, CGA, EGA, or VGA.

Display Modes

The following *ioctl*s can be used to change the video display mode:

SW_B80x25	Selects 80x25 black and white text display mode. (MONO, CGA, EGA, VGA)
SW_C80x25	Selects 80x25 color text display mode. (CGA, EGA, VGA)
SW_B40x25	Selects 40x25 black and white text display mode. (MONO, CGA, EGA, VGA)
SW_C40x25	Selects 40x25 color text display mode. (CGA, EGA, VGA)
SW_BG320	Selects 320x200 black and white graphics display mode. (CGA, EGA, VGA)
SW_CG320	Selects 320x200 color graphics display mode. (CGA, EGA, VGA)
SW_BG640	Selects 640x200 black and white graphics display mode. (CGA, EGA, VGA)
SW_EGAMONO80x25	Selects EGA (Enhanced Graphics Adapter) mode 7 - emulates support provided by the monochrome display. (EGA, VGA)

SW_EGAMONOAPA	Selects EGA support for 640x350 graphics display mode (EGA mode F). (EGA with mono monitor)
SW_ENHMONOAPA2	Selects EGA mode F*. (EGA with mono monitor)
SW_ENHB40x25	Selects enhanced EGA support for 40x25 black and white text display mode. (EGA, VGA)
SW_ENHC40x25	Selects enhanced EGA support for the 40x25 color text display mode. (EGA, VGA)
SW_ENHB80x25	Selects enhanced EGA support for 80x25 black and white text display mode. (EGA, VGA)
SW_ENHC80x25	Selects enhanced EGA support for 80x25 color text display mode. (EGA, VGA)
SW_ENHB80x43	Selects enhanced EGA support for 80x43 black and white text display mode. (EGA, VGA)
SW_ENHC80x43	Selects enhanced EGA support for 80x43 color text display mode. (EGA, VGA)
SW_CG320_D	Selects EGA support for 320x200 graphics display mode. (EGA mode D.) (EGA, VGA)
SW_CG640_E	Selects EGA support for 640x200 graphics display mode (EGA mode E). (EGA, VGA)
SW_CG640x350	Selects EGA support for 640x350 graphics display mode (EGA mode 10). (EGA, VGA)
SW_ENH_CG640	Selects EGA mode 10*. (EGA, VGA)
SW_MCAMODE	Reinitializes the monochrome adapter. (MONO)
SW_VGA40x25	Selects VGA support for the 40x25 color text display mode (VGA mode 1+). (VGA)
SW_VGA80x25	Selects VGA support for the 80x25 black and white text display mode (VGA mode 2+). (VGA)

SW_VGAM80x25	Selects VGA mode 7+ - emulates support provided by the monochrome display. (VGA with mono monitor)
SW_VGA11	Selects VGA support for the 640x480 graphics display mode (VGA mode 11). (VGA)
SW_VGA12	Selects VGA support for the 640x480 graphics display mode (VGA mode 12). (VGA)
SW_VGA13	Selects VGA support for the 320x200 graphics display mode (VGA mode 13). (VGA)

Switching to an invalid display mode for a display device will result in an error.

Getting Display Modes

The following *ioctl( )* requests are provided to obtain information about the current display modes:

CONS_GET	Returns the current display mode setting for current display adapter. (All)
CGA_GET	Returns the current display mode setting of the color graphics adapter. (CGA only)
EGA_GET	Returns the current display mode setting of the enhanced graphics adapter. (EGA only)
MCA_GET	Returns the current display mode setting of the monochrome adapter. (MONO only)
VGA_GET	Returns the current display mode of the video graphics adapters. (VGA only)
CONS_GETINFO	Returns structure *vid_info* (below). Size of structure (first field) must be filled in by user.

```
struct vid_info
{
        short   size;                   /* must be first field          */
        short   m_num;                  /* multiscreen number, 0 based  */
        ushort  mv_row, mv_col;         /* cursor position              */
        ushort  mv_rsz, mv_csz;         /* text screen size             */
        struct  colors mv_norm,         /* normal attributes            */
                       mv_rev,          /* reverse video attributes     */
                       mv_grfc;         /* graphic character attributes */
        uchar_t mv_ovscan;              /* border color                 */
        uchar_t mk_keylock;             /* caps/num/scroll lock         */
};
```

CONS_6845INFO — Returns structure *m6845_info* (below). Size of structure (first field) must be filled in by user.

```
struct m6845_info
{
    short   size;          /* must be first field        */
    ushort  screen_top;    /* offset of screen in video  */
    ushort  cursor_type;   /* cursor shape               */
};
```

CONSADP — Returns number of multiscreen displayed on adaptor associated with that multiscreen.

GIO_ATTR — Return value of ioctl is 6845-style attribute byte in effect.

GIO_COLOR — Return value of ioctl is zero or one depending on whether the device supports color.

GIO_SCRNMAP — Gets the 256-byte screen map table, which is the mapping of ASCII values (0-256) onto the PC video ROM font characters (0-256). Note that control characters (ASCII values less than hex 20) have control functions and do not display ROM characters (example: ^J is newline).

This is often used to map the low font values that normally correspond to ASCII control values to higher ASCII values, thus displaying the desired ROM characters.

PIO_SCRNMAP — Puts the 256-byte screen map table (see GIO_SCRNMAP).

PIO_KEYMAP — See *keyboard*(HW)

PIO_KEYMAP — See *keyboard*(HW)

GIO_FONT8X*n* — Gets font, where *n* is 8, 14, and 16. Argument is a pointer to a font table. Size of 8X8 font table is 8X256 bytes, 8X14 is 14X256 bytes, etc.

PIO_FONT8X*n* — Puts font, where *n* is 8, 14, and 16. Argument is a pointer to a font table. Size of 8X8 font table is 8X256 bytes, 8X14 is 14X256 bytes, etc.

Memory Mapping Modes

The *ioctl* (S) routine is used to map the display memory of the various devices into the user's data space.

Note that the MAP* ioctls map the memory associated with the current mode. You must put the adapter into the desired mode before performing mapping, or the pointers returned will not be appropriate. Refer to your hardware manual for details on various displays, adapters, and controllers.

These *ioctl* ( ) requests can be used to map the display memory:

MAPCONS	Maps the display memory of the adaptor currently being used into the user's data space. (All)
MAPMONO	Maps the monochrome adapter's display memory into the user's data space. (MONO only)
MAPCGA	Maps the color adapter's display memory into the user's data space. (CGA only)
MAPEGA	Maps the enhanced graphics adapter's display memory into the user's data space. (EGA only)
MAPVGA	Maps the video graphics adapter's display memory into the user's data space. (VGA only)

For example, the following code can be used to acquire a pointer to the start of the user data space associated with the color graphics adapter display memory:

```
char *dp;
int retval;
     .
     .
     .
/* fd is a file descriptor for a
   multiscreen device */
retval = ioctl (fd, MAPCONS,0L);
dp = (char *) retval;
     .
     .
     .
```

Note that when the display memory is mapped into the user space, the adapter's m6845 start address registers are not set. The start address can be reset in two ways, so that the start address of the display memory corresponds to the upper left hand corner of the screen:

1. Switch modes with an *ioctl*() (the "switch" can be to the present mode). See the "Display Modes" section of this manual page.

2. Change the start address high and low address with the *in-on-port/out-on-port ioctl*().

The *in-on-port/out-on-port* ioctl()'s can also be used to determine the current value in the start address register, and then set up a pointer to point to the offset in the mapped-in data space.

MAP_CLASS
Package ioctl that gives I/O privileges to an arbitrary list of ports and maps an arbitrary frame buffer into user's address space identified by a string found in the struct vidclass vidclasslist[] located in **/etc/conf/pack.o/class.h**.

KDDISPTYPE
This call returns display information to the user. The argument expected is the buffer address of a structure of type *kd_disparam* into which display information is returned to the user. The *kd_disparam* structure is defined as follows:

```
struct kd_disparam {
        long type;      /*display type*/
        char *addr;     /*display memory address*/
        ushort ioaddr[MKDIOADDR];       /*valid I/O addresses*/
}
```

Possible values for the *type* field include:

KD_MONO (0x01), for the IBM monochrome display adapter.

KD_HERCULES (0x02), for the Hercules monochrome graphics adapter.

KD_CGA (0x03), for the IBM color graphics adapter.

KD_EGA (0x04), for the IBM enhanced graphics adapter.

KD_VGA (0x05), for the IBM video graphics adapter.

KDDISPINFO
Returns *struct kd_disparam*, which contains adaptor type and physical address of frame buffer.

KIOCSOUND
Start sound generation. Turn on sound. The *arg* is the frequency desired. A frequency of 0 turns off the sound. This is useful for generating tones while in graphics mode.

KDGETLED
Get keyboard LED status. The argument is a pointer to a character. The character will be filled with a boolean combination of the following values:

1 scroll lock
2 num lock
4 caps lock

KDSETLED
Set keyboard LED status. The argument is a character whose value is the boolean combination of the values listed under "KDGETLED".

KDMKTONE
Not supported. (See KIOCSOUND.)

KDADDIO
Not supported. (See MAP_CLASS.)

KDDELIO
Not supported. (See MAP_CLASS.)

KIOCDOSMODE
Not supported.

KIOCNONDOSMODE
Not supported.

KDSETMODE
(VP/IX only.) Set console in text or graphics mode. The argument is of type integer, which should contain one of the following values:

KD_TEXT	0x00	(sets console to text mode)
KD_GRAPHICS	0x01	(sets console in graphics mode)

Note, the user is responsible for programming the color/graphics adaptor registers for the appropriate graphical state.

KDGETMODE
(VP/IX only.) Get current mode of console. Returns integer argument containing either KD_TEXT or KD_GRAPHICS as defined in the KDSETMODE ioctl description.

KDENABIO
Enable in's and out's to video adaptor ports. No argument.

KDDISABIO
Disable in's and out's to video adaptor ports. No argument.

KDGKBTYPE
Always returns 0.

KDGKBMODE
Get keyboard translation mode, also known as scan code mode. Mode is returned where arg points.

KDSKBMODE
Set keyboard translation mode, also known as scan code mode.

KDGKBSTATE
Returns the state of the shifted, alt-, or control- state of the keyboard. Returns a bollean combination of:
1 shifted
2 control-
4 alt-

KIOCINFO
Always returns 0x6664.

KDMAPDISP
(VP/ix only) Maps display memory into user process address space. Argument is a pointer to structure type *kd_memloc*. This ioctl requires that a virtual 8086 subtask be attached to the current process. KDMAPDISP should not be used by ordinary users to map the console display; use MAPCONS.

KDUNMAPDISP
(VP/ix only) Unmap display memory from user process address space. No argument required.

VT_SETMODE
Set the virtual terminal mode. The argument is a pointer to a vt_mode structure, as defined below.

VT_GETMODE
Determine what mode the active virtual terminal is currently in, either VT_AUTO or VT_PROCESS. The argument to the ioctl is the address of the following type of structure:

```
struct vt_mode {
char  mode; /* VT mode */
      char    waitv;  /* not implemented */
      short   relsig; /* signal to use for release request */
      short   acqsig; /* signal to use for display acquired */
      short   frsig;  /* not implemented */
      }

      #define VT_AUTO         0x00    /* automatic VT switching */
      #define VT_PROCESS      0x01    /* process controls switching */
```

The vt_mode structure will be filled in with the current value for each field.

VT_RELDISP
Used to tell the virtual terminal manager that the display has or has not been released by the process.

0 == release refused
1 == release acknowledged
2 == acquire acknowledged

VT_ACTIVATE
Makes the multiscreen number specified in the argument the active multiscreen. The video driver will cause a switch to occur in the same manner as if a hotkey sequence had been typed at the keyboard. If the specified multiscreen is not open or does not exist, the call will fail and errno will be set to ENXIO.

Graphics Adapter Port I/O

You can use *ioctl*(S) to read or write a byte from or to the graphics adapter port. The *arg* parameter of the *ioctl* call uses the *io_arg* data structure:

```
struct port_io_arg {
     struct port_io_struct args[4];
};
```

As shown above, the *io_arg* structure points to an array of four *port_io* data structures. The *port_io* structure has the following format:

```
struct port_io_struct {
        char dir;     /* direction flag (in vs. out) */
        unsigned short port;    /* port address */
        char data;    /* byte of data */
};
```

You may specify one, two, three, or four of the *port_io_struct* structures in the array for one *ioctl* call. The value of *dir* can be either IN_ON_PORT to specify a byte being input to the graphics adapter port or OUT_ON_PORT to specify a byte being output to the graphics adapter port. *Port* is an integer specifying the port address of the desired graphics adapter port. *Data* is the byte of data being input or output as specified by the call.

If you are not using any of the *port_io* structures, load the *port* with 0, and leave the unused structures at the end of the array. Refer to the hardware manuals for port addresses and functions for the various adapters.

You can use the following *ioctl(S)* commands to input or output a byte on the graphics adapter port:

CONSIO	Inputs or outputs a byte on the current graphics adapter port as specified. (All)
MGAIO	Inputs or outputs a byte on the monochrome adapter port as specified. (MONO only)
CGAIO	Inputs or outputs a byte on the color graphics adapter port as specified. (CGA only)
EGAIO	Inputs or outputs a byte on the enhanced graphics adapter port as specified. (EGA only)
VGAIO	Inputs or outputs a byte on the video graphics array adapter port as specified. (VGA only)

To input a byte on any of the graphics adapter ports, load *dir* with IN_ON_PORT and load *port* with the port address of the graphics adapter. The byte input from the graphics adapter port will be returned in *data*.

To output a byte, load *dir* with OUT_ON_PORT, load *port* with the port address of the graphics adapter, and load *data* with the byte you want output to the graphics adapter port.

Function Keys

ioctl(S) can be used to define or obtain the current definition of a function key. The **arg** parameter of the **ioctl** call uses the **fkeyarg** data structure:

```
struct fkeyarg {
        unassigned int keynum;
        char keydef [MAXFK];
        /* Comes from
        char flen; ioctl.h via comcrt.h */
}
```

You can use the following **ioctl**(S) requests to obtain or assign function key definitions:

GETFKEY	Obtains the current definition of a function key. The function key number must be passed in **keynum**. The string currently assigned to the key will be returned in **keydef** and the length of the string will be returned in **flen** when the **ioctl** is performed.

SETFKEY	Assigns a given string to a function key. The function key number must be passed in **keydef** and the length of the string (number of characters) must be passed in **flen**.
SETLOCKLOCK	Toggles the <Caps Lock> and <Num Lock> keys to be either global to all the multiscreens, or local to each individual multiscreen. To make the <Caps Lock> global (its default), set the *arg* parameter to 1. To make the <Caps Lock> local to each screen, set the *arg* parameter to 0.

ANSI Screen Attribute Sequences

The following character sequences are defined by ANSI X3.64-1979 and may be used to control and modify the screen display. Each **n** is replaced by the appropriate ASCII number (decimal) to produce the desired effect. The last column is for *termcap* (M) codes, where "n/a" means not applicable.

The use of 7 or 8 bit characters in the escape sequence is a valid invocation for each action defined. For example the ANSI ED command can be invoked via the "ESC [*n* J" (0x1b-0x5b-*n*-0x4a, 7 bit chars) sequence or the "CSI *n* J" (0x9b-*n*-0x4n, 8 bit chars) sequence.

ISO	Sequence	Action	Termcap Code
ED (Erase in Display)	CSI *n* J	Erases all or part of a display. *n=0*: erases from active position to end of display. *n=1*: erases from the beginning of display to active position. *n=2*: erases entire display.	cd
EL (Erase in Line)	CSI *n* K	Erases all or part of a line. *n=0*: erases from active position to end of line. *n=1*: erases from beginning of line to active position. *n=2*: erases entire line.	ce
ECH (Erase Character)	CSI *n* X	Erases *n* characters	n/a

CBT (Cursor Backward Tabulation)	CSI*n* Z	Moves active position back *n* tab stops.	bt
SU (Scroll Up)	CSI*n* S	Scroll screen up *n* lines, introducing new blank lines at bottom.	sf
SD (Scroll Down)	CSI*n* T	Scrolls screen down *n* lines, introducing new blank lines at top.	sr
CUP (Cursor Position)	CSIm;*n* H	Moves active position to location *m* (vertical) and *n* (horizontal).	cm
HVP (Horizontal & Vertical Position)	CSIm;*n* f	Moves active position to location *m* (vertical) and *n* (horizontal).	n/a
CUU (Cursor Up)	CSI*n* A	Moves active position up *n* number of lines.	up (ku)
CUD (Cursor Down)	CSI*n* B	Moves active position down *n* number of lines.	do (kd)
CUF (Cursor Forward)	CSI*n* C	Moves active position *n* spaces to the right.	nd (kr)
CUB (Cursor Backward)	CSI*n* D	Moves active position *n* spaces backward.	bs (kl)
HPA (Horizontal Position Absolute)	CSI*n* ‘	Moves active position to column given by *n*.	n/a
HPR (Horizontal Position Relative)	CSI*n* a	Moves active position *n* characters to the right.	n/a

SCREEN (HW)

VPA (Vertical Position Absolute)	CSI *n* d	Moves active position to line given by *n*.	n/a
VPR (Vertical Position Relative)	CSI *n* e	Moves active position down *n* number of lines.	n/a
IL (Insert Line)	CSI *n* L	Inserts *n* new, blank lines.	al
ICH (Insert Character)	CSI *n* @	Inserts *n* blank places for **n** characters.	ic
DL (Delete Line)	CSI *n* M	Deletes *n* lines.	dl
DCH (Delete Character)	CSI *n* P	Deletes *n* number of characters.	dc
CPL (Cursor to Previous Line)	CSI *n* F	Moves active position to beginning of line, *n* lines up.	n/a
CNL (Cursor Next Line)	CSI *n* E	Moves active position to beginning of line, *n* lines down.	n/a

SGR (Select Graphic Rendition)	CSI *n* m	Character attributes, as summarized in the chart below. Multiple attributes can be specified in the form: CSI *n1*; *n2*; *n3* m	n/a

Select Graphic Rendition (SGR) Chart

n	Meaning	
0	all attributes off (normal display)	
1	bold intensity (or light color)	
4	underscore on (if hardware supports it)	
5	blink on (if hardware supports it)	
7	reverse video	
8	sets blank (non-display)	
10	selects the primary font	
11	selects the first alternate font; lets ASCII characters less than 32 be displayed as ROM characters	
12	selects a second alternate font; toggles high bit of extended ASCII code before displaying as ROM characters	
30	black	foreground
31	red	foreground
32	green	foreground
33	brown	foreground
34	blue	foreground
35	magenta	foreground
36	cyan	foreground
37	white	foreground
38	enables underline option; white foreground with white underscore	
39	disables underline option	
40	black	background
41	red	background
42	green	background
43	brown	background
44	blue	background
45	magenta	background
46	cyan	background
47	white	background

ISO	Sequence	Action	Termcap Code
SM (Set Mode)	CSI [2 h	Lock keyboard. Ignores keyboard input until unlocked. Characters are not saved.	n/a

MC (Media Copy)	CSI[2i	Send screen to host. Current screen contents are sent to the application.	n/a
RM (Reset Mode)	CSI[21	Unlock keyboard. Re-enable keyboard input.	n/a

Additional Screen Attribute Sequences

Name	Sequence	Action	Termcap Code
n/a	CSI=*p*;*d*B	Set the bell parameter to the decimal values of *p* and *d*. *p* is the period of the bell tone in units of 840.3 nanoseconds, and *d* is the duration of the tone in units of 100 milliseconds.	n/a
n/a	CSI=*s*;*e*C	Set the cursor to start on scanline *s* and end on scanline *e*.	n/a
n/a	CSI=*x*D	Turn on or off (*x*=1 or 0) the intensity of the background color.	n/a
n/a	CSI=*x*E	Set or clear (*x*=1 or 0) the Blink vs. Bold background bit in the 6845 crt controller.	n/a
n/a	CSI=*c*A	Set overscan color to color *c*. *c* is a decimal value taken from Color Table above. (This sequence may not be supported on all hardware.)	n/a
n/a	CSI=*c*F	Set normal foreground color to *c*. (*c* is a decimal parameter taken from Color Table.)	n/a
n/a	CSI=*c0G*	Set normal background. (See Color Table.)	n/a
n/a	CSI=*c* H	Set reverse foreground. (See Color Table.)	n/a

n/a	CSI = *c* I	Set reverse background. (See Color Table.)	n/a
n/a	CSI = *c* J	Set graphic foreground. (See Color Table.)	n/a
n/a	CSI = *c* K	Set graphic background. (See Color Table.)	n/a

Color Table

Cn	Color	Cn	Color
0	Black	8	Grey
1	Blue	9	Lt. Blue
2	Green	10	Lt. Green
3	Cyan	11	Lt. Cyan
4	Red	12	Lt. Red
5	Magenta	13	Lt. Magenta
6	Brown	14	Yellow
7	White	15	Lt. White

Name	Sequence	Action	Termcap Code
n/a	CSI [*n* g	Accesses alternate graphics set. Not the same as "graphics mode." Refer to your owner's manual for decimal/character codes (**Pn**) and possible output characters.	n/a
n/a	ESC Q Fn '*string*'	Define function key **Fn** with *string*. String delimiters ' and ' may be any character not in *string*. **Fn** is defined as the key number starting at zero plus the ASCII value of zero. For example, **F1** = 0... **F16** = ?, and so on. In this escape sequence, the ^ character will cause the next character to have 32 subtracted from its ASCII value. Thus ^! results in a soh (^A) character.	n/a
n/a	CSI *n* z	*n* should be equal to the number of the screen to switch to. If screen does not exist, no action will take place.	n/a

Files

/dev/console

/dev/tty [02 -*n*]

/dev/color

/dev/monochrome

/dev/ega

/dev/vga

See Also

console(M), ioctl(S), keyboard(HW), keymap(M), mapkey(M), mapchan(M), multiscreen(M), setcolor(C), stty(C), systty(M), vidi(C), termcap(M), tty(M)

scsi

small computer systems interface

Description

SCSI provides a standard interface for peripherals such as hard disks, printers, tape drives and others. SCSI is run via a host adapter card that can support up to 8 controllers, each supporting up to 8 devices. Note that only 4 hard disks can be attached to a controller.

The minor device numbering scheme for SCSI disk devices is the same as the standard minor device number scheme for non-SCSI disk devices. The minor device numbering scheme for SCSI tape devices is as follows:

SCSI Tape Minor Devices

Bits								
7	6	5	4	3	2	1	0	Description
X	X	-	-	-	-	-	-	Unit (LUN)
-	-	X	-	-	-	-	-	No unload on close
-	-	-	X	-	-	-	-	High density (6250 BPI)
-	-	-	-	X	-	-	-	No rewind on close
-	-	-	-	-	X	X	X	Unused(reserved)

Each SCSI controller has its own major device number.

See Also

hd(HW), tape(HW)

Value Added

scsi is an extension of AT&T System V provided by the Santa Cruz Operation.

serial: tty1[a-h] , tty1[A-H] , tty2[a-h] , tty2[A-H]

interface to serial ports

Description

The **tty1[a-h]**, **tty1[A-H]**, **tty2[a-h]** and **tty2[A-H]** files provide access to the standard and optional serial ports of the computer. Each file corresponds to one of the serial ports (with or without modem control). Files are named according to the following conventions:

- The first number in the file name corresponds to the COM expansion slot.
- Lower case letters indicate no modem control.
- Upper case letters indicate the line has modem control.

tty1a and **tty1A** both refer to COM 1, whereas **tty2a** and **tty2A** both refer to COM 2.

For example, with a four port expansion board installed at COM 1 and a single port board installed at COM 2, you can access:

tty1a	tty1A
tty1b	tty1B
tty1c	tty1C
tty1d	tty1D
tty2a	tty2A

Each serial port has modem and non-modem invocations. The device names in the following table refer to the serial ports, with and without modem control. The first section of the table describes boards at COM 1 and the second section describes boards installed at COM 2. "Minor" is the minor device number for the port (see *mknod*(C)).

SERIAL (HW)

Serial Lines				
Board Type	Non-Modem Control		Modem Control	
	Minor	*Name*	*Minor*	*Name*
1 Port	0	tty1a	128	tty1A
4 Port	1	tty1b	129	tty1B
	2	tty1c	130	tty1C
	3	tty1d	131	tty1D
8 Port	4	tty1e	132	tty1E
	5	tty1f	133	tty1F
	6	tty1g	134	tty1G
	7	tty1h	135	tty1H
1 Port	8	tty2a	136	tty2A
4 Port	9	tty2b	137	tty2B
	10	tty2c	138	tty2C
	11	tty2d	139	tty2D
8 Port	12	tty2e	140	tty2E
	13	tty2f	141	tty2F
	14	tty2g	142	tty2G
	15	tty2h	143	tty2H

Interrupt Vectors:

All board(s) installed at COM 1 - 4
All board(s) installed at COM 2 - 3

For a list of I/O addresses, see the *Release Notes* furnished with your distribution.

Access

The files may only be accessed if the corresponding serial interface card is installed and its jumper I/O address correctly set. Also, for multi-port expansion cards, you must use the *mkdev*(ADM) program to create more than the default number of files. Unless other COM slots are specifically referred to in your hardware documentation, only COM 1 and COM 2 may be used.

The serial ports must also be defined in the system configuration. Check your hardware manual to determine how your system is configured, via a CMOS database or by switch settings on the main system board. If your system is configured using a CMOS database, the ports are defined in the database (see *cmos*(HW)). Otherwise, define the ports by setting the proper switches on the main system board. Refer

to your computer hardware manual for switch settings.

It is an error to attempt to access a serial port that has not been installed and defined.

The serial ports can be used for a variety of serial communication purposes such as connecting login terminals to the computer, attaching printers, or forming a serial network with other computers. Note that a serial port may operate at most of the standard baud rates, and that the ports (on most computers) have a DTE (Data Terminal Equipment) configuration. The following table defines how each pin is used for 25-pin and 9-pin connections:

25-Pin	9-Pin	Description
2	2	Transmit Data
3	3	Receive Data
4	7	Request to Send
5	8	Clear to Send
7	5	Signal Ground
8	1	Carrier Detect (Data Set Ready)
20	4	Data Terminal Ready

Only pins 2, 3, and 7 (2,3 and 5 for 9-pin) are necessary for a terminal (or direct) connection.

A modem control device (port) uses pins 2, 3, and 7 in the same way as a non-modem control device: send on pin 2 and receive on pin 3. Pin 7 is data ground. On a non-modem control device, pins 4 and 20 (RTS and DTR) are asserted, but pin 8 is not. On a modem control device, pins 4 and 20 (RTS & DTR) are asserted and the port will not open until pin 8 (CXD) is asserted. That is, no signal travels from pin 2 until pin 8 is asserted from another source. The modem control device monitors the the status of pin 8.

See *tty*(M) and *termio*(M) for the details of serial line operation on UNIX systems.

Files

/dev/tty1[a-h]
/dev/tty1[A-H]
/dev/tty2[a-h]
/dev/tty2[A-H]

See Also

cmos(HW), csh(C), cu(C), getty(ADM), mkdev(ADM), mknod(C) nohup(C), open(S), termio(M), tty(M), uucp(C)

Notes

If you login via a modem control serial line, hanging up logs that line out and kills your background processes. See *nohup*(C) and *csh*(C)).

You cannot use the same serial port with both modem and non-modem control at the same time. For example, you cannot use tty1a and tty1A simultaneously.

Use a modem cable to connect your modem to a computer.

TAPE (HW)

tape

magnetic tape device

Description

The *tape* device implements the UNIX interface with a tape drive. QIC-02 cartridge tape drives are supported by the *ct* device driver, QIC-40 and QIC-80 tape drives connected to the floppy disk controller are supported with the *ft* device driver, and Irwin tape drives connected to the floppy disk controller are supported with the *mc* device driver. Typically, the *tar*(C), *cpio*(C), *dd*(C), *backup*(ADM), *xbackup*(ADM), *xrestore*(ADM), or *restore*(ADM) commands are used to access a tape drive.

A single tape drive with a raw (character, non-blocking) interface is supported, except for the SCSI tape driver which supports up to four devices. There are four standard tape device types. Devices beginning with the "r" prefix, (for "raw device"), should be used for most normal tape work, while devices with the "n" prefix, ("for no rewind on hold"), should be used for storing and restoring multiple files. Devices beginning with the "x" prefix are control devices, which are used for sending *ioctl*(S) commands to the tape subsystem.

Devices beginning with the "e" prefix (for ECC device) support a 2/64 error recovery scheme. Thus two 512-byte blocks out of every 64 blocks can be bad and the driver will correct the errors. This software ECC support provides a high degree of error recovery.

The *ft* and *mc* floppy tape drivers do not support the "n" or "e" device types. ECC encoding and decoding is automatically used with the standard "r" device. On the QIC-40, QIC-80 and Irwin 80MB drives, for every 29K written to the tape, 3K of ECC data is written with it to provide error recovery. On the Irwin 10, 20, 40 and 60MB drives, for every 16K written to the tape, 2K of ECC data is written.

QIC-40 and QIC-80 tapes must be formatted with the *tape*(C) command before use, unless you use pre-formatted tapes. Similarly, Irwin tapes must be first servo-written and then formatted with *tape*(C) before use, unless you use pre-formatted tapes. The *mc* driver can read tapes formatted and written under XENIX but cannot write to them.

The following table summarizes the base naming conventions for the tape drives supported:

ct0,1	QIC24 unit 0,1
ct2,3	QIC11 unit 0,1
Stp0,1,2,3	SCSI tape unit 0,1,2,3

ft0	QIC-40 or QIC-80 floppy tape unit
mc0	Irwin floppy tape unit
ctmini	default mini-cartridge device
mt0,1	reel to reel unit 0,1 1600 bpi
mt2,3	reel to reel unit 0,1 800 bpi
mt4,5	reel to reel unit 0,1 6250 bpi

The default tape device is stored in the file **/etc/default/tape**, which is also used by *tape*(C). **/etc/default/tape** should always contain the "x" (control) device name of the default device, and is normally updated by *mkdev*(ADM) *tape*. If the default device is a QIC-40, QIC-80 or Irwin tape drive, the appropriate device from the table above will be linked to the *ctmini* device node. QIC-02 tape drives will always be accessed by the *ct0,1* device nodes as shown in the table. If a SCSI tape drive is installed as the default device and there is no QIC-02 drive installed, it will be linked to the *ct0* device node. If both SCSI and QIC-02 drives are installed, the SCSI device node cannot be linked to the *ct0* device node.

tape(C) describes the commands used to access tape drives.

Definition of ioctl commands

The following *ioctl* commands can be used with the various tape device drivers supported under UNIX. The letters following each description indicate which drivers support each *ioctl* command:

A	All drivers
C	QIC-02 cartridge tape driver
S	SCSI tape driver
F	QIC-40 and QIC-80 mini-cartridge tape drivers
I	Irwin mini-cartridge tape driver

MT_STATUS
Returns a device-independent structure holding the status of the drive. The *tape_info* structure is defined in **/usr/include/sys/tape.h**. (C,S,F)

MT_DSTATUS
Returns a device-dependent structure holding status information of the drive. (C,S,F)

MT_RESET
Resets the driver software and the tape drive. Interrupts tape commands in progress. (C,S,F)

MT_REPORT
Returns an integer code which determines the type of device which the driver controls. The type numbers are defined in **/usr/include/sys/tape.h**. (C,S,F)

MT_RETEN
Winds the tape forward to EOT and then backward to BOT. (C,S,F)

MT_REWIND
Rewinds the tape to BOT. (C,S,F)

MT_ERASE
Erases the data on the tape and retensions the cartridge. (C,S,F)

MT_AMOUNT
Returns an integer count of the amount of the last data transfer. (C,S,F)

MT_FORMAT
Formats the tape. Expects as an argument the number of tracks to format, which must be an even number. If no argument is provided, the default is 20 tracks for QIC-40 drives, and 28 tracks for QIC-80 drives. (F)

MT_GETHDR
Expects as an argument a pointer to a *struct ft_header* or *struct ir_header* and copies the header of the current tape into it. (F)

MT_PUTHDR
Takes a pointer to a *struct ft_header* or *struct ir_header* and writes it onto the tape. This command should be used with caution. (F)

MT_GETNEWBB
Takes a pointer to a *struct ft_newbbt* or *struct ir_newbbt* and copies in a list of bad blocks detected on the last write operation. (F)

MT_PUTNEWBB
Takes a pointer to a *struct ft_newbbt* or *struct ir_newbbt*, reads in the header from the tape, then writes a new bad block onto the tape with the new bad blocks from the provided bad block table. (F)

MT_GETVTBL
Takes a pointer to a *struct ft_vtbl* and copies in the volume table from the tape. (F)

MT_PUTVTBL
Takes a pointer to a *struct ft_vtbl* and writes the volume table onto the tape. This command should be used with caution. (F)

MT_RFM
Winds the tape forward to the next file mark. (C,S)

MT_WFM
Writes a file mark at the current location on the tape. (C,S)

MT_LOAD
On devices which are capable of doing so, loads the tape into the drive. (S)

MT_UNLOAD
On devices which are capable of doing so, unloads the tape from the drive. (S)

Irwin-specific ioctl Interface

Device specific functions of the Irwin tape drive are accessed via special commands passed to the Irwin driver using the *ioctl()* interface. An Irwin driver interface library is available. This library provides a system independent interface to *ioctl()* via the entry point *mcioctl()*:

```
#include "mc.h"

int mcioctl(fh, cmd, arg)
int fh;         /* File handle from open() */
int cmd;        /* MCCTL_* command code */
void *arg;      /* Additional argument pointer */

mcioctl(fh, MCCTL_NOP, NULL)
mcioctl(fh, MCCTL_VERSION, verbuf)
mcioctl(fh, MCCTL_CAPACITY, capp)
mcioctl(fh, MCCTL_LSEEK, lskbuf);
mcioctl(fh, MCCTL_REWIND)
mcioctl(fh, MCCTL_RETEN)
mcioctl(fh, MCCTL_REWIND_NW)
mcioctl(fh, MCCTL_RETEN_NW)
mcioctl(fh, MCCTL_GETDRVCFG, cfgbuf)
mcioctl(fh, MCCTL_GETCFG, cfgbuf)
mcioctl(fh, MCCTL_SETCFG, cfgbuf)
mcioctl(fh, MCCTL_GETTHDR, hdrbuf)
mcioctl(fh, MCCTL_PUTTHDR, hdrbuf)
mcioctl(fh, MCCTL_GETDLISTS, listbuf)
mcioctl(fh, MCCTL_FLUSH)
mcioctl(fh, MCCTL_FORMAT, fmtbuf)
mcioctl(fh, MCCTL_FMTSTAT, fmtbuf)
mcioctl(fh, MCCTL_ABORT)
mcioctl(fh, MCCTL_DEVSTAT, dstatp)
mcioctl(fh, MCCTL_GETERCTL, erctlp)
mcioctl(fh, MCCTL_SETERCTL, erctlp)
mcioctl(fh, MCCTL_GETER, ierrp)
struct mcver *verbuf;     /* version buffer */
long *capp;               /* capacity in bytes */
struct mclseek *lskbuf;   /* tape logical position descriptor */
struct mccfg *cfgbuf;     /* configuration buffer */
char *hdrbuf;             /* 1024 byte header buffer */
unsigned short * listbuf; /* 2048 byte defect list buffer */
struct mcfmt *fmtbuf;     /* format control/status buffer */
unsigned short *dstatp;   /* device status word */
unsigned short *erctlp;   /* error control word */
unsigned short *ierrp;    /* device specific error */
```

mcioctl() provides system independent ioctl interface to the Irwin driver. This subroutine is essentially a pass-through. That is, arguments are passed through to *ioctl()*. If a device specific error occurs (i.e., a non-system error) at completion of the system *ioctl()* and the command is other than MCCTL_NOP or MCCTL_VERSION, *mcioctl()* executes *ioctl(MCCTL_GETER)* to retrieve the device specific error.

The following ioctl commands are available for the Irwin driver:

MCCTL_NOP
No operation. The argument is ignored. A success status is returned. This command may be used as an aid in determining if a special file refers to the MC driver.

MCCTL_VERSION
Gets driver version information. The argument is the address of version information buffer (see *struct mcver* in **/usr/include/sys/mc.h**) to which the driver writes.

MCCTL_CAPACITY
Gets a tape's capacity in bytes. The argument is the address of a long integer.

MCCTL_REWIND
MCCTL_RETEN
MCCTL_REWIND_NW
MCCTL_RETEN_NW
These four commands physically position the tape at high speed. MCCTL_RETEN and MCCTL_RETEN_NW run the tape to the early warning hole first. All four commands return the tape to the load-point hole. MCCTL_REWIND_NW and MCCTL_RETEN_NW start a request but don't wait for completion.

MCCTL_GETDRVCFG
MCCTL_GETCFG
MCCTL_SETCFG
These three commands provide access to configuration parameters for a particular mini cartridge tape unit. The structure of these parameters is *struct mccfg* (defined in **/usr/include/sys/mc.h**) This structure has driver, tape drive, and cartridge related fields. Both MCCTL_GETDRVCFG and MCCTL_GETCFG copies the driver's the MCCFG structure to the caller's buffer. When MCCTL_GETDRVCFG is used, *struct mccfg* members with driver and tape drive related fields are returned. No error is given when a cartridge is absent. When MCCTL_GETCFG is used successfully, all fields are returned with valid data. An error is returned if no cartridge is present. MCCTL_SETCFG allows the caller to adjust certain fields in the driver's configuration.

MCCTL_GETTHDR
MCCTL_PUTTHDR

MCCTL_GETTHDR and MCCTL_PUTTHDR read and write the 1024 byte tape header in block 0. MCCTL_PUTTHDR assumes an Irwin style header. The the following procedure is used to write the header:

Tape block 0 is read to a buffer. The caller's 1024 byte header buffer is copied to the first, fifth, and when space permits, the ninth and thirteenth 1024-byte sectors in the buffer. When the cartridge format uses ECC (i.e., other than 110 cartridge format), the header's ECC in use field is set. When the cartridge format uses ECC, ECC is encoded. A check sum is calculated for the buffer. The buffer is written back to block 0. Block 0 is reread and the cartridge state is redetermined. A new checksum is calculated and compared against the original.

MCCTL_GETDLISTS

Returns lists used by the driver's flaw management. The caller gives the address of a buffer which is at least 2 KB in length. Four lists are copied to the buffer. Each list is comprised of physical tape block numbers stored as unsigned short integers and terminated with the value 0xffff. The lists are contiguous and given in the following order:

Primary Defect List (PDL)
Working Defect List (WDL)
Grown Defect List (GDL)
Relocation List (RL)

MCCTL_FLUSH

Flushes dirty buffers to tape. MCCTL_FLUSH forces dirty buffers in the Irwin driver's cache to be written to tape. The pointer argument is ignored. Control returns when data is written. Buffers are automatically flushed upon a *close()* or when the device is idle for a certain period (see *mc_autoflush* in *struct mccfg* in **/usr/include/sys/mc.h**).

MCCTL_FORMAT
MCCTL_FMTSTAT

MCCTL_FORMAT starts a erase, servo-format-certify-initialize header or re-certify operations. The argument is the address of *struct mcfmt* (see **/usr/include/sys/mc.h**). Formatting operations performed depend upon the values in the structure's *fm_cmd* and *fm_option* fields, and *struct mccfg mc_cartstate field*. When an MCCTL_FORMAT command completes successfully, MCCTL_FMTSTAT is used to determine the progress (when a no-wait flag is set) or results of formatting. Like MCCTL_FORMAT, MCCTL_FMTSTAT also uses the *struct mcfmt* structure (typically the same one passed to MCCTL_FORMAT).

MCCTL_ABORT
Used to interrupt and terminate operations started by MCCTL_FORMAT. The pointer argument is ignored. Control returns after formatting has terminated.

MCCTL_DEVSTAT
Returns a 16-bit device status word to an unsigned short integer who's address is passed in the third argument of ioctl(). This field is intended for use by applications which use the tape drive interactively. The status bits are defined in *struct mclseek* in **/usr/include/sys/mc.h**.

MCCTL_GETERCTL
MCCTL_SETERCTL
MCCTL_GETERCTL and MCCTL_SETERCTL give application access to the state of and control over certain error mechanisms. The argument is the address of a 16-bit error control variable which the Irwin driver writes with current values for MCCTL_GETERCTL and reads for MCCTL_SETERCTL. Certain flags may or may not have an effect depending on the implementation. Bit values for the error control variable are defined in **/usr/include/sys/mc.h**.

MCCTL_GETER
Gets device specific error: IE_*. In general the value 0 is returned to indicate success or -1 to indicate an error. When *mcioctl()* returns the value -1, an error has occurred. The error condition may have been detected in the operating system or in the driver. In order to discriminate the origin the global *_mcerrno* should be examined first (before errno). When non-zero, the error was returned by the driver. Values for *_mcerrno* are defined in **/usr/include/ierrno.h** with an IE_ prefix.

Irwin Drive and Cartridge Models

This section is concerned with Irwin tape drives and cartridges supported.

Drive Models

Many Irwin mini cartridge drives have a three digit model number. Each digit has a meaning. The high order digit encodes the form factor and cabinetry:

1xx	5-1/4 inch drive (mounted in system cabinet).
2xx	3-1/2 inch drive (mounted in system cabinet).
3xx	5-1/4 inch drive in a metal cabinet w/ power supply.
4xx	3-1/2 inch drive in a plastic cabinet (no supply).
7xx	3-1/2 inch drive in a metal cabinet w/ power supply.

The middle digit gives the approximate capacity, in 10 Megabyte units for a standard capacity (not extra long) tape:

x1x	10 Megabytes
x2x	20
x4x	40
x6x	60
x8x	80

The low digit encodes the drive's normal data transfer rate (i.e., the floppy controller data clock rate).

xx0	250 Kilobits/Second
xx5	500 Kilobits/Second
xx7	1 Megabit/Second

In addition, a new 4-digit model numbering system is in use. These model numbers are associated with drives which are adaptable to different system hardware environments with accessory hardware kits.

2020	3-1/2 inch, 20 Megabyte, 250 Kilobits/Second
2040	3-1/2 inch, 40 Megabyte, 500 Kilobits/Second
2080	3-1/2 inch, 80/120 Megabyte, 500 Kilobits/Second
2120	3-1/2 inch, 80/120 Megabyte, 1 Megabit/Second

Mini Cartridges

There are three primary physical mini cartridges types:

DC1000	185 feet of 0.150 inch wide tape (same as TC-200)
DC2000	205 feet of 0.250 inch wide tape (same as TC-400)
DC2120	307.5 feet of 0.250 inch wide tape

TAPE (HW)

The DC1000 cartridge is physically thinner than DC2000 and DC2120 cartridges. The DC2000 and DC2120 have the same physical form but the DC2120 has a longer tape. These cartridges are distinguished by their labels. Each physical cartridge type has at least two cartridge formats:

Mini (Irwin) Cartridge Format Parameters

Cartridge Format	AccuTrak Reorder Number see note	Cartridge	Total Tape Blocks	Trks	Blocks per Track	Sectors per Block		Density (FTPI)
						Data	ECC	
110	1000-10	DC1000	1264	8	158	8	0	6400
120	2000-20	DC2000	1190	14	85	16	2	6400
120XL	2000-30	DC2120	1792	14	128	16	2	6400
125	1000-20	DC1000	1320	12	110	16	2	10000
145	2000-40	DC2000	2480	20	124	16	2	10000
145XL	2000-60	DC2120	3720	20	186	16	2	10000
165	2000-64	DC2000	3936	24	164	16	2	13200
285	2000-80	DC2000	2752	32	86	29	3	11600
285XL	2000-120	DC2120	4160	32	130	29	3	11600

Notes: The suffix part of the AccuTrak Reorder Number is an approximate cartridge capacity in Megabytes.

All formats use 1024 byte MFM encoded sectors.

Drive Read/Write Compatibility for Mini Cartridge Formats

	Drive Model (See Note)							
Cartridge Format	410 310 110	2020 720 420 320 220 120	725 425 325 225 125	2040 745 445 345 245 145	765 465 265 165	2080 785 485 285	2120 787 487 287	Cartridge
110	rw	rw	r-	r-	r-	r-	r-	DC1000
120	--	rw	--	r-	r-	r-	r-	DC2000
120XL	--	rw	--	r-	r-	r-	r-	DC2120
125	--	--	rw	rw	r-	r-	r-	DC1000
145	--	--	--	rw	r-	r-	r-	DC2000
145XL	--	--	--	rw	r-	r-	r-	DC2120
165	--	--	--	--	rw	r-	r-	DC2000
285	--	--	--	--	--	rw	rw	DC2000
285XL	--	--	--	--	--	rw	rw	DC2120

Key: r Drive reads cartridge format

w Drive writes cartridge format

\- Incompatible: When a cartridge is formatted but incompatible for reading or writing, the driver reports that the cartridge is either incompatible or erased.

Extra Long (XL) DC2120 Cartridge Compatibility

Extra long (i.e., DC2120) cartridges are incompatible with the following drives as the drive will not physically accommodate the cartridge: 110, 310, 410, 125, 225, 325, 425, and 725

Even though DC2120 cartridges are physically accepted in the following drives, they may not be formattable:

120, 220, 320, 420, 720, 2020, 145, 245, 345, 445, 745, 2040

Drives manufactured previous to about 1989 don't recognize the longer tape. However, the MC driver is able to read and write preformated extra long tapes in these drives, but it is unable to correctly format them. Formatting will start, but terminate in error. To determine whether a drive supports formatting of DC2120 cartridges, use the mcart utility. If the command *mcart drive* reports a drive type with the suffix XL, formatting of DC2120 cartridges is supported.

Files

/dev/rStp0	/dev/rct0	/dev/erct0	/dev/rmc1
/dev/nrStp0	/dev/nrct0	/dev/xct0	/dev/mcdaemon
/dev/xStp0	/dev/rct2	/dev/rctmini	
/dev/rft0	/dev/nrct2	/dev/xctmini	
/dev/xft0	/dev/xct0	/dev/rmc0	

Include files:

/usr/include/sys/tape.h	/usr/include/sys/ir.h
/usr/include/sys/ct.h	/usr/include/sys/mc.h
/usr/include/sys/ft.h	/usr/include/sys/mcheader.h

Notes

After certain tape operations are executed, the system returns a prompt before the tape controller has finished its operation. If the user enters another tape command too quickly, a "device busy" error is returned until the tape device is finished with its previous operation.

Periodic tape cartridge retensioning and tape head cleaning are necessary for continued error-free operation of the tape subsystem. Use *tape*(C) to retension the tape.

See Also

backup(ADM), xbackup(ADM), cpio(C), dd(C), format(C), tape(C), tar(C), restore(ADM), xrestore(ADM)

terminal

login terminal

Description

A *terminal* is any device used to enter and display data. It may be connected to the computer:

- By a serial wire, either direct or dialup
- As a virtual terminal, for example with emulator software
- Through a display adapter

A terminal has an associated device file **/dev/tty***.

Files

/dev/tty*

See Also

console(M), disable(C), enable(C), mkdev(ADM), serial(HW), stty(C), vidi(C), termcap(M), term(F), terminals(M)

XT (HW)

xt

multiplexed tty driver for AT&T windowing terminals

Description

The *xt* driver provides virtual *tty*(M) circuits multiplexed onto real *tty*(M) lines. It interposes its own channel multiplexing protocol as a line discipline between the real device driver and the standard *tty*(M) line disciplines.

Virtual *tty*(M) circuits are named by character-special files of the form */dev/xt???*. File names end in three digits, where the first two represent the channel group and the last represents the virtual *tty*(M) number (0-7) of the channel group. Allocation of a new channel group is done dynamically by attempting to open a name ending in **0** with the **O_EXCL** flag set. After a successful open, the *tty*(M) file onto which the channels are to be multiplexed should be passed to *xt* via the **XTIOCLINK** *ioctl*(S) request. Afterwards, all the channels in the group will behave as normal *tty*(M) files, with data passed in packets via the real *tty*(M) line.

The *xt* driver implements the protocol described in *xtproto*(M) and in *layers*(M). Packets are formatted as described in *xtproto*(M), while the contents of packets conform to the description in *layers*(M).

There are three groups of *ioctl*(S) requests recognized by *xt*. The first group contains all the normal tty *ioctl*(S) requests described in *termio*(M), with the addition of the following:

TIOCEXCL Set exclusive use mode; no further opens are permitted until the file has been closed.

TIOCNXCL Reset exclusive use mode; further opens are once again permitted.

The second group of *ioctl*(S) requests concerns control of the windowing terminal, and is described in the header file **<sys/jioctl.h>**. The requests are as follows:

JTYPE, JMPX Both return the value **JMPX.** These are used to identify a terminal device as an *xt* channel.

JBOOT, JTERM Both generate an appropriate command packet to the windowing terminal affecting the layer associated with the file descriptor argument to *ioctl*(S). They may return the error code **EIO** if the system *clist* is empty.

JTIMO, JTIMOM **JTIMO** specifies the timeouts in seconds, and **JTIMOM** in milliseconds. Invalid except on channel 0. They may return the error code **EIO** if the system *clist* is empty.

JWINSIZE Requires the address of a *jwinsize* structure as an argument. The window sizes of the layer associated with the file descriptor argument to *ioctl*(S) are copied to the structure.

JZOMBOOT Generate a command packet to the windowing terminal to enter download mode on the channel associated with the file descriptor argument to *ioctl*(S), like **JBOOT**; but when the download is finished, make the layer a zombie (ready for debugging). It may return the error code **EIO** if the system *clist* is empty.

JAGENT Send the supplied data as a command packet to invoke a windowing terminal agent routine, and return the terminal's response to the calling process. Invalid except on the file descriptor for channel 0. See *jagent*(M). It may return the error code **EIO** if the system *clist* is empty.

The third group of *ioctl*(S) requests concerns the configuration of *xt*, and is described in the header file **<sys/xt.h>**. The requests are as follows:

XTIOCTYPE Returns the value **XTIOCTYPE.**

XTIOCLINK Requires an argument that is a structure, *xtioclm*, containing a file descriptor for the file to be multiplexed and the maximum number of channels allowed. Invalid except on channel 0. This request may return one of the following errors:

EINVAL *nchans* has an illegal value.

ENOTTY *fd* does not describe a real *tty*(M) device.

ENXIO *linesw* is not configured with *xt*.

EBUSY An **XTIOCLINK** request has already been issued for the channel group.

ENOMEM There is no system memory available for allocating to the *tty*(HW) structures.

EIO The **JTIMOM** packet described above could not be delivered.

HXTIOCLINK Like **XTIOCLINK,** but specifies that ENCODING MODE be used.

XTIOCTRACE Requires the address of a *tbuf* structure as an argument. The structure is filled with the contents of the driver trace buffer. Tracing is enabled. This request is invalid if tracing is not configured.

XTIOCNOTRACE Tracing is disabled. This request is invalid if tracing is not configured.

XTIOCSTATS Requires an argument that is the address of an array of size S_NSTATS, of type *Stats_t*. The array is filled with the contents of the driver statistics array. This request is invalid if statistics are not configured.

XTIOCDATA Requires the address of a maximum-sized *Link* structure as an argument. The structure is filled with the contents of the driver *Link* data. This request is invalid if data extraction is not configured.

Files

/dev/xt/??[0-7]	multiplexed special files
/usr/include/sys/jioctl.h	packet command types
/usr/include/sys/xtproto.h	channel multiplexing protocol definitions
/usr/include/sys/xt.h	driver specific definitions

See Also

layers(C), termio(M), tty(M), ioctl(S), open(S), libwindows(S), jagent(M), layers(M)

512-210-950
23027